CHAPTER 3

Blocking API

This module describes the features and operations of the Blocking API and provides code examples.

This module 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 the Nonblocking API module.

Information About Multithreading 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, page 3-30.

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

Multithreading 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 Figure 3-1 (the vertical scale is time):

Figure 3-1 Multithreading Support

The SM allocates five threads to handle each API instance. It is recommended to develop a multithreaded application that uses the API with a number of threads in the order of the five threads. Implementing with more threads might result in longer delays for the calling threads.

Operation Timeout Error Code

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

The reply timeout feature, the **setReplyTimeout** method, lets the caller set a timeout. It will return a ReturnCode with the **ERROR_CODE_CLIENT_OPERATION_TIMEOUT** error when a reply does not return within the timeout period.

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

Information About Blocking API Methods

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

The Blocking API is a superset of the Nonblocking API. Except for differences in return values and result handling, identical operations in both APIs have the same functions and syntax structure.

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, page 3-4
  - logoutByName, page 3-7
  - logoutByNameFromDomain, page 3-8
  - logoutByMapping, page 3-10
  - loginCable, page 3-11
  - logoutCable, page 3-13

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, page 3-14
  - removeSubscriber, page 3-16
  - removeAllSubscribers, page 3-17
  - setPropertiesToDefault, page 3-28
  - removeCustomProperties, page 3-29
- For simple read-only operations performed independently in subscriber awareness mode, the following methods are relevant:
  - getNumberOfSubscribers, page 3-17
  - getNumberOfSubscribersInDomain, page 3-18
  - getSubscriber, page 3-19
  - subscriberExists, page 3-20
  - subscriberLoggedIn, page 3-21
  - getSubscriberNameByMapping, page 3-22
  - getSubscriberNames, page 3-23
  - getSubscriberNamesInDomain, page 3-25
  - getSubscriberNamesWithPrefix, page 3-26
  - getSubscriberNamesWithSuffix, page 3-26
  - getDomains, page 3-27

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, page 3-30
  - C++ init Method, page 3-30
  - C SMB_init Function, page 3-31
  - C SMB_release Function, page 3-32

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

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, page 2-8.

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, page 2-10.

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, page A-1.

Example

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

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

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

To add the IP address 192.168.12.5 overriding previous mappings:

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

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

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

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

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

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

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

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

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

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

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

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

logoutByName

Syntax

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

Description

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

Parameters

argName—See explanation of Subscriber Name Format, page 2-8.

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, page A-1.

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

```
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, page 2-8.

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, page 2-10.

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, page A-1.

Example

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

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

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

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

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

logoutByMapping

Syntax

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

Description

Locates a subscriber based on domain and mapping, and removes the 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, page A-1.

Example

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

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

loginCable

Syntax

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

Description

A login method adapted for the cable environment, 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 Service Control Management Suite Subscriber Manager User Guide*.

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

Parameters

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, page 2-10.

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 "Default Domains Configuration" 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

```
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

```
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, page 2-8.

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, page 2-10.

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, page A-1.

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

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

Description

Removes a subscriber completely from the SM database.

Parameters

argName—See explanation of Subscriber Name Format, page 2-8.

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, page A-1.

Example

To remove subscriber *john* entirely from the database:

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

removeAllSubscribers

Syntax

```
ReturnCode* removeAllSubscribers()
```

Description

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

Note This method 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

```
ReturnCode* getNumberOfSubscribers()
```

Description

Retrieves the total number of subscribers in the SM database.

Return Value

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

Error Codes

None.

getNumberOfSubscribersInDomain

Syntax

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

Description

Retrieves the number of subscribers in a subscriber domain.

Parameters

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

Return Value

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

Error Codes

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

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE _DOMAIN_NOT_FOUND

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

getSubscriber

Syntax

```
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, page 2-8.

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, page A-1.

Example

To retrieve the subscriber record of *john*:

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

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

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

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

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

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

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

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

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

subscriberExists

Syntax

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

Description

Verifies that a subscriber exists in the SM database.

Parameters

argName—See explanation of Subscriber Name Format, page 2-8.

Return Value

A pointer to a ReturnCode structure with a Boolean type:

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

Error Codes

None.

subscriberLoggedIn

Syntax

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

Description

Checks whether a subscriber that already exists in the SM database is logged in; 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, page 2-8.

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, page A-1.

getSubscriberNameByMapping

Syntax

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

Description

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

Parameters

argMapping—See explanation of mappings and mapping types 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, page A-1.

getSubscriberNames

Syntax

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

Description

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

If **argFirstName** is NULL, the (alphabetically) first subscriber name that exists in the SM database will be used.

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

Parameters

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

argAmount—Limit on 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, page A-1.

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

```
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, page A-1.

getSubscriberNamesWithPrefix

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, page A-1.

getSubscriberNamesWithSuffix

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, page A-1.

getDomains

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

```
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, page 2-8.

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, page A-1.

removeCustomProperties

Syntax

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

Description

Resets the specified custom properties of a subscriber.

Parameters

argName—See explanation of Subscriber Name Format, page 2-8.

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, page A-1.

C++ setLogger Method

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

```
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 wanteddelay 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

```
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

```
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

```
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

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

Description

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

Parameters

argName—The API name.

Return Value

None.

connect

Syntax

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

Description

Attempts to establish a PRPC protocol connection to the SM.

Parameters

argHostName—The SM IP-Address or hostname.

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

Return Value

Boolean value:

- TRUE—Success
- FALSE—Fail

disconnect

Syntax

```
bool disconnect()
```

Description

Attempts to terminate the PRPC protocol connection to the SM.

Return Value

Boolean value:

- TRUE—Success
- FALSE—Fail

isConnected

Syntax

```
bool isConnected();
```

Description

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

Return Value

Boolean value:

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

Blocking API C++ Code Examples

This section provides two code examples:

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

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);
    }
}
```