Cisco SCMS SM C/C++ API プログラマ ガイド

Release 3.1
May 2007

Text Part Number: OL-7203-05-J

CONTENTS

このマニュアルについて

May 30, 2007, OL-7203-05-J

このマニュアルでは、SCMS SM C/C++ API の概要と、そのインストール、コンパイル、実行の手順について説明します。

SCMS SM C/C++ API は、Subscriber Manager（SM）のアップデート、クエリ、設定に使用されます。SCMS SM C/C++ API は次に示す 2 つの部分で構成されています。これらは、個別に使用しても併せて使用してもかまいません。

- SM ノンブロッキング C/C++ API：エラーやその他の操作結果に対する可視性が低い高性能 API。OSS/AAA システムとの自動統合をサポートしています。
- SM ブロッキング C/C++ API：ユーザ フレンドリな API。SM のアクセスおよび管理用のユーザ インターフェイス アプリケーションをサポートしています。

（注） まったく同じ機能を持つ API セットが Java 環境でも利用できます。

この概要では、次のトピックについて説明します。

対象読者

このマニュアルは、SM の設定を担当するネットワーク技術者またはコンピュータ技術者、および SCE プラットフォームを管理するオペレータを対象としています。

マニュアルの変更履歴

Cisco Service Control リリース	Part Number	発行日
Release 3.1.0	OL-7203-05	2007 年 5 月

変更点

- サブスクライバのドメイン間移動をサポート。login メソッドの「サブスクライバ ドメイン」（p.2-11）および「パラメータ」（p.3-6）を参照してください。
- 「サブスクライバ名のフォーマット」（p.2-9）の更新

Cisco Service Control リリース	Part Number	発行日
Release 3.0.5	OL-7203-04	2006 年 11 月

変更点

- C++ init メソッドに reconnect timeout パラメータを追加。「C++ init メソッド」（p.3-32）を参照してください。
- setReconnectTimeout メソッドを追加。「setReconnectTimeout」（p.3-34）を参照してください。

Cisco Service Control リリース	Part Number	発行日
Release 3.0.3	OL-7203-03	2006 年 5 月

変更点

- Release 3.0.3 のマニュアル更新

Cisco Service Control リリース	Part Number	発行日
Release 3.0	OL-7203-02	2005 年 12 月

変更点

- マニュアルの構成変更。このリビジョンでは主な変更や新機能の追加はありません。

Cisco Service Control リリース	Part Number	発行日
Release 2.5.7	OL-7203-01	2005 年 5 月

変更点

- このマニュアルの初版です。

マニュアルの構成

このマニュアルの構成は、次のとおりです。

表 1　マニュアルの構成

章	タイトル	説明
第 1 章	使用する前に	C/C++ API を使用できるプラットフォームと、C/C++ API コンポーネントのインストール、コンパイル、および起動の手順について説明します。
第 2 章	API の基礎知識	SM C/C++ API を使用する際に役立つさまざまな事項について説明します。
第 3 章	ブロッキング API	ブロッキング API の機能と動作について説明し、コードの例を紹介します。
第 4 章	ノンブロッキング API	ノンブロッキング API の機能と動作について説明し、コードの例を紹介します。
付録 A	エラー コードのリスト	C/C++ API で使用されるエラー コードのリストを紹介します。

表記法

このマニュアルでは、次の表記法を使用しています。

- **太字**は、コマンド、キーワード、およびボタンを表します。
- *イタリック体*は、ユーザによる値の指定が必要なコマンド入力を表します。
- `screen` フォントは、画面に表示される情報の例を表します。
- **太字**の **`screen`** フォントは、ユーザが入力する情報の例を表します。
- 縦棒（|）は相互に排他的な要素を区切るときに使用します。
- 角カッコ（[ ]）は省略可能な要素を表します。
- 波カッコ（{ }）は必須の選択肢を表します。
- 波カッコが角カッコで囲まれている（[{ }]）場合は、省略可能な要素における必須の選択肢を表します。

（注）　「*注釈*」です。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。

ワンポイント・アドバイス　「*時間の節約に役立つ操作*」です。記述されている操作を実行すると時間を節約することができます。

注意　「*要注意*」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。

警告　**「*危険*」の意味です。人身事故を予防するための注意事項が記述されています。機器の取り扱い作業を行うときは、電気回路の危険性に注意し、一般的な事故防止対策に留意してください。**

関連資料

このマニュアル『*Cisco SCMS SM C/C++ API プログラマ ガイド*』は、SCMS Subscriber Manager のユーザ ガイド、API ガイド、リファレンス ガイドと併せてご利用ください。

マニュアルの入手方法、テクニカル サポート、およびシスコのセキュリティ ガイドライン

マニュアルの入手方法、テクニカル サポート、マニュアルに関するフィードバックの提供、セキュリティ ガイドライン、推奨されるエイリアス、および一般的なシスコのマニュアルに関する情報については、次の URL で、毎月更新される『*What's New in Cisco Product Documentation*』を参照してください。『*What's New in Cisco Product Documentation*』にはシスコの新規および改訂版の技術マニュアルの一覧が示されています。

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

Japan TAC Web サイト

Japan TAC Web サイトでは、利用頻度の高い TAC Web サイト（http://www.cisco.com/tac）のドキュメントを日本語で提供しています。Japan TAC Web サイトには、次の URL からアクセスしてください。

http://www.cisco.com/jp/go/tac

サポート契約を結んでいない方は、「ゲスト」としてご登録いただくだけで、Japan TAC Web サイトのドキュメントにアクセスできます。

Japan TAC Web サイトにアクセスするには、Cisco.com のログイン ID とパスワードが必要です。ログイン ID とパスワードを取得していない場合は、次の URL にアクセスして登録手続きを行ってください。

http://www.cisco.com/jp/register/

CHAPTER 1

使用する前に

この章では、使用可能なプラットフォーム、インストール、コンパイル、および起動方法の面から SM C/C++ API について説明します。

プラットフォームとコンパイラ

SCMS SM C/C++ API は、Windows、Solaris、Linux のプラットフォーム上で開発とテストが行われました。Windows では Microsoft Visual C++ 6 コンパイラ、Solaris では GCC 2.95.3 コンパイラ、Linux では GCC 3.2.3 コンパイラでコンパイルされています。

インストール パッケージの内容

ここでは便宜上、インストール ディレクトリ **sm-c-api-vvv.bb** を **<installdir>** と表します。**vvv** と **bb** はそれぞれ SCMS SM C/C++ API のバージョン番号とビルド番号を表します。

<installdir>/winnt フォルダには、**smapi.dll** ファイルが入っています。これは、Windows API 実行可能ファイルです。さらに、Windows API の実行に必要なその他の DLL および LIB ファイルもこのフォルダに入っています。

<installdir>/solaris フォルダには、**libsmapi.so** ファイルが入っています。これは、Solaris API 実行可能ファイルです。

<installdir>/linux フォルダには、**libsmapi.so** ファイルが入っています。これは、Linux API 実行可能ファイルです。

<installdir>/include フォルダには、API C/C++ ヘッダー ファイルが入っています。

<installdir>/include/system フォルダには、C++ API 内部ヘッダー ファイルが入っています。

表 1-1　インストール ディレクトリのレイアウト

フォルダ	サブフォルダ（該当する場合）	ファイル名
<installdir>		README.csmapi
	include	BasicTypes.h
		Common.h
		GeneralDefs.h
		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

SCMS SM C/C++ API のインストール方法

SCMS SM C/C++ API ディストリビューションは SCMS SM-LEG 配布ファイルの一部であり、sm_api ディレクトリに入っています。SCMS SM C/C++ API は UNIX tar ファイルに圧縮されています。SCMS SM C/C++ API は UNIX tar ユーティリティまたはほとんどの Windows 圧縮ユーティリティで解凍できます。

- UNIX プラットフォーム、および Linux プラットフォームへのディストリビューションのインストール（p.1-3）
- Windows へのディストリビューションのインストール（p.1-3）

UNIX プラットフォーム、および Linux プラットフォームへのディストリビューションのインストール

1. SCMS SM-LEG 配布ファイルを解凍します。
2. C/C++ SM API の配布ファイル **sm-c-api-dist.tar** の場所を確認します。
3. C/C++ API パッケージ tar を解凍します。

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

Windows へのディストリビューションのインストール

1. zip 解凍ユーティリティ（WinZip など）を使用してパッケージを解凍します。
2. インストレーション ウィザードで表示されるプロンプトに従って手順を実施します。

SCMS SM C/C++ API のコンパイルおよび実行方法

API は SM 上の PRPC サーバに接続します。API を機能させるためには、次の条件を満たす必要があります。

- SM が起動され稼働中であり、API のホスト マシンから到達できること。
- SM 上の PRPC サーバが起動されていること。

PRPC サーバはシスコが開発した独自の RPC プロトコルです。詳細については、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。

次の各セクションでは、それぞれサポートされるソフトウェア プラットフォームで API をコンパイルする方法、および実行する方法について説明します。

- Windows におけるプログラムのコンパイルと実行（p.1-4）
- Solaris におけるプログラムのコンパイルと実行（p.1-5）
- Red Hat Linux におけるプログラムのコンパイルと実行（p.1-5）

Windows におけるプログラムのコンパイルと実行

手順の概要

1. **smapi.dll** およびその他の DLL ファイルがパスまたは実行可能ファイルのディレクトリにあることを確認します。
2. **include** フォルダがコンパイルのインクルード パスにあることを確認します。
3. smapi.lib ファイルがリンケージ パスにあることを確認します。
4. 該当する API ヘッダー ファイルをソース コードに含めて、コードをコンパイルします。

手順の詳細

ステップ 1　**smapi.dll** およびその他の DLL ファイルがパスまたは実行可能ファイルのディレクトリにあることを確認します。

ステップ 2　**include** フォルダがコンパイルのインクルード パスにあることを確認します。

Microsoft Visual C++ 6 を使用する場合の例：

プロジェクト設定値を入力し、**C++** タブをクリックしてから、**Preprocessor** カテゴリを選択します。インクルード ディレクトリのパスを **Additional Include directories** 行に追加します。

ステップ 3　smapi.lib ファイルがリンケージ パスにあることを確認します。

Microsoft Visual C++ 6 を使用する場合の例：

プロジェクトの設定値を入力し、**Link** タブをクリックします。**smapi.lib** を **Object\Library modules** 行に追加します。

ステップ 4　該当する API ヘッダー ファイルをソース コードに含めて、コードをコンパイルします。

Solaris におけるプログラムのコンパイルと実行

手順の概要

1. **libsmapi.so** が LD_LIBRARY_PATH にあることを確認します。
2. **include** フォルダがコンパイルのインクルード パスにあることを確認します。
3. **libsmapi.so** ファイルをリンケージ行に入れるか、またはこれを動的にロードします。

手順の詳細

ステップ 1　**libsmapi.so** が LD_LIBRARY_PATH にあることを確認します。

たとえば、**Bash** シェル タイプを使用している場合は、次のコマンドラインを使用します。

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

ステップ 2　**include** フォルダがコンパイルのインクルード パスにあることを確認します。

たとえば、**GCC** を使用している場合は、次のように **-I** オプション フラグの後ろにインクルード フォルダを追加します。

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

ステップ 3　**libsmapi.so** ファイルをリンケージ行に入れるか、またはこれを動的にロードします。

次のように、オブジェクト ファイルを *pthread* と socket ライブラリにリンクします。

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

Red Hat Linux におけるプログラムのコンパイルと実行

手順の概要

1. **libsmapi.so** が LD_LIBRARY_PATH にあることを確認します。
2. **include** フォルダがコンパイルのインクルード パスにあることを確認します。
3. **libsmapi.so** ファイルをリンケージ行に入れるか、またはこれを動的にロードします。

手順の詳細

ステップ 1　**libsmapi.so** が LD_LIBRARY_PATH にあることを確認します。

たとえば、**Bash** シェル タイプを使用している場合は、次のコマンドラインでプロンプトに応じてパスワードを入力します。

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

ステップ 2　**include** フォルダがコンパイルのインクルード パスにあることを確認します。

たとえば、**GCC** を使用している場合は、次のように **-I** オプション フラグの後ろにインクルード フォルダを追加します。

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

ステップ 3　**libsmapi.so** ファイルをリンケージ行に入れるか、またはこれを動的にロードします。

-L オプション フラグを使用して **libsmapi.so** の場所を指定します。次のように、オブジェクト ファイルを *pthread* と stdc++ ライブラリにリンクします。

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

CHAPTER 2

API の基礎知識

この章では、SM C/C++ API を使用する際に役立つさまざまな事項について説明します。

ブロッキング API とノンブロッキング API

ここでは、ブロッキング API とノンブロッキング API の操作の相違点を説明します。

ブロッキング API

ブロッキング API は最も一般的な API です。ブロッキング API では、いずれのメソッドも操作が*完了したあと*で制御が戻ります。

SM ブロッキング C/C++ API には広範な操作があり、ノンブロッキング API 機能の*上位集合*に相当します。

ノンブロッキング API

ノンブロッキング API のメソッドは、操作が完了していなくても*即座に*戻ります。操作結果は、ユーザが定義したコールバック セットに戻るか、またはまったく戻りません。

ノンブロッキング メソッドは、入出力が含まれていて時間のかかる操作の場合に有利です。別のスレッドで操作を実行すれば、呼び出し側はほかのタスクを続行できるので、システム全体のパフォーマンスが向上します。

SM ノンブロッキング C/C++ API には、ノンブロッキング操作がいくつか含まれています。この API は、結果コールバックによる操作結果の取得をサポートしています。

信頼性

SCMS SM C/C++ API は、API 上で実行された操作が、SM から対応する結果が届くまで維持されるという意味では信頼性が高いといえます。SM への接続が切断された場合、SM に送信されなかった操作と SM からまだ結果が届いていない操作は、再接続後ただちに SM に送信されます。操作呼び出しの順序は常に維持されます。

C と C++ API

C と C++ の API は、基本的には同じものです。C API は、実際には C++ API のシン ラッパーにすぎず、違いはオブジェクト指向のプログラミング言語ではない C の制約によるメソッド プロトタイプやシグニチャの相違だけです。

次に、C と C++ API の相違点を説明し、いくつかの例を紹介します。

- メソッド名（p.2-3）
- ハンドル ポインタ（p.2-3）

メソッド名

C API のメソッド名には識別のためのプレフィクスが付きますが、それ以外の点では C API と C++ API のメソッド名は同じです。

- ブロッキング C API のメソッド名には、SMB_ というプレフィクスが付きます。
- ノンブロッキング C API のメソッド名には、SMNB_ というプレフィクスが付きます。

（注） このマニュアルでは、C++ API のメソッドの名前とシグニチャが使用されています。

例：

ブロッキングとノンブロッキングの C++ API にはいずれも **login** メソッドがあります。同じメソッドがブロッキング C API では **SMB_login**、ノンブロッキング C API では **SMNB_login** になります。

ハンドル ポインタ

ブロッキング API およびノンブロッキング API は、C++ オブジェクトです。複数の API インスタンスが 1 つのプロセス内に共存し、それぞれのインスタンスにソケットとステートがあります。C API では同じ機能を提供するためにハンドルという概念が導入されています。C API の各メソッドはその最初の引数としてハンドルを受け入れます。ハンドルは呼び出し側が操作の対象とする C API インスタンスを識別するために使用されます。**SMB_init** メソッドまたは **SMNB_init** メソッドを呼び出すと新しい C API インスタンスが作成され、戻り値にインスタンスのハンドルが含まれます。詳細については、「API の初期化」（p.2-5）を参照してください。

- ブロッキング API の例（p.2-3）
- ノンブロッキング API の例（p.2-4）

ブロッキング API の例

次に C++ ブロッキング API の **logoutByName** メソッドのシグニチャを示します。

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

C ブロッキング API では次のようになります。

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

ノンブロッキング API の例

次に C++ ノンブロッキング API の **logoutByName** メソッドのシグニチャを示します。

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

C ノンブロッキング API では次のようになります。

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

API の初期化

API の初期化は次の手順で行います。

- 2 つのコンストラクタの 1 つを使用し、API を構築します。
- API 固有のセットアップ操作を実行します。
- API を SM に接続します。

次の各セクションでは、これらの手順について詳しく説明します。

- API の構築（p.2-5）
- セットアップ操作（p.2-6）
- SM への接続（p.2-6）

初期化の例は、各 API のコード例を参照してください。

API の構築

C および C++ のブロッキングおよびノンブロッキング API では、API の構築と初期化を行う必要があります。手順を進める前に、初期化が正常に終了していることを確認してください。

C++ API を構築、および初期化するには、次の例のように、まず API オブジェクトを構築してから、**init** 関数を呼び出します。

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

C API を構築、初期化する場合は、**init** 関数を呼び出します。この関数によって API の割り当てと初期化が実行されます。

例：

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

LEG 名の設定

SM で SM-LEG 障害ハンドリング オプションを有効にする場合は、Login Event Generator（LEG）名を設定します。LEG と SM-LEG 障害ハンドリングの詳細については、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。

LEG 名を設定するには、API の該当する **setName** 関数を呼び出します。SM は、接続障害から回復するときに LEG 名を使用します。LEG 名には、次のように API を識別する文字列定数が付加されます。

- ブロッキング API：**.B.SM-API.C**
- ノンブロッキング API：**.NB.SM-API.C**

LEG 名の設定例（ブロッキング API）

設定した LEG 名が **my-leg.10.1.12.45-version-1.0** の場合、実際の LEG 名は **my-leg.10.1.12.45-version-1.0.B.SM-API.C** になります。

LEG 名を設定しないと、名前のプレフィクスとして、そのマシンのホスト名が使用されます。

SM-LEG 障害ハンドリングの詳細については、『*Cisco SCMS Subscriber Manager User Guide*』の「Appendix A」を参照してください。

セットアップ操作

セットアップ操作は、2 つの API で異なります。いずれの API でも切断リスナの設定がサポートされます。詳細については、「切断コールバック リスナ」（p.2-13）を参照してください。

次に、ブロッキング API とノンブロッキング API のセットアップ操作について説明します。

- ブロッキング API のセットアップ（p.2-6）
- ノンブロッキング API のセットアップ（p.2-6）

ブロッキング API のセットアップ

ブロッキング API をセットアップするには、操作タイムアウト値を設定する必要があります。詳細については、「ブロッキング API」（p.3-1）を参照してください。

ノンブロッキング API のセットアップ

切断コールバックを設定する必要のあるノンブロッキング API をセットアップする場合は、「ノンブロッキング API」（p.4-1）を参照してください。

SM への接続

次の例は、C++ API 使用時の SM への接続方法を示しています。

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

次の例は、C ブロッキング API 使用時の接続方法を示しています。

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

次の例は、C ノンブロッキング API 使用時の接続方法を示しています。

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

argHostName パラメータは、IP アドレスまたは到達可能なホスト名のいずれかになります。**isConnected** 関数のいずれか 1 つのバリアントを使用することにより、API 操作中にいつでも、API が接続されているかどうかを確認できます。

API の終了

C と C++ のブロッキング API およびノンブロッキング API はいずれも SM から切断し、API のメモリを解放する必要があります。

- C++ API の場合は、**disconnect** メソッドを呼び出し、API オブジェクトを解放します。
- C API の場合は、適切な **disconnect** 関数を呼び出してから、適切な **release** 関数を使用して API を解放します。

リターン コード構造体

API 操作の結果は、**ReturnCode** と呼ばれる包括的構造体を使用して戻されます。**ReturnCode** 構造体には、次のようなパラメータが含まれています。

- **u** ― 実際の戻り値となる変数および変数へのポインタすべての共用体
- **type** ― ReturnCode 構造体が保有する値（**u**）の型を定義するリターン コード型パラメータ（**ReturnCodeType** 列挙体）
- **size** ― 値内の要素の数。**size** が 1 である場合、スカラ、文字列、void、エラーなど値の要素は 1 つです。**size** が 1 よりも大きい場合は、配列内にいくつかの要素があり、その型は配列型の 1 つになります。

リターン コード構造体は API によって割り当てられますが、解放は API ユーザが行う必要があります。構造体を安全に解放するには、**freeReturnCode** ユーティリティ関数を使用します。

その他のリターン コード構造体ユーティリティ関数は次のとおりです。

- **printReturnCode** ― stdout に **ReturnCode** 構造体値を出力します。
- **isReturnCodeError** ― **ReturnCode** 構造体がエラーかどうかを調べます。

定義

次に **GeneralDefs.h** ヘッダー ファイルからの定義を示します。

```
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;
```

リターン コード構造体の例

次の例では、*subscriber1* というサブスクライバ データが取得され、表示されます。戻ってきた構造体には、**objectArray** 共用体値に格納された ReturnCode 構造体の配列が含まれています。各構造体には、さまざまな型の値が含まれています。

詳細は、「getSubscriber」メソッドの説明を参照してください。次のコード例では、**isReturnCodeError** および **freeReturnCode** の各メソッドが使用されています。

```
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);
```

エラー コード構造体

リターン コード構造体の値の 1 つとして、**ErrorCode** 構造体が返ってくることがあります。この構造体は、発生したエラーに関する情報を示しています。この構造体は次のパラメータで構成されています。

- **type** ― エラーを記述する ErrorCodeType 列挙体。詳細は **GeneralDefs.h** ファイルを参照してください。
- **message** ― 具体的なエラー コード
- **name** ― 現在は未使用

エラー コード構造体の定義

次に **GeneralDefs.h** ヘッダー ファイルからの定義を示します。

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

サブスクライバ名のフォーマット

いずれの API も、ほとんどのメソッドは入力パラメータとしてサブスクライバ名を必要とします。ここでは、サブスクライバ名のフォーマット ルールについて説明します。

使用できる文字数は最大 64 文字です。ASCII コード 32 ～ 126 （126 も含む）の範囲の印刷可能文字を使用できます。ただし 34（"）、39（'）、および 96（`）は除きます。

ネットワーク ID マッピングについて

ネットワーク ID マッピングは、SCE デバイスが特定のサブスクライバ レコードと関連付けることのできるネットワーク識別子です。たとえば IP アドレスは、ネットワーク ID マッピング（または単純にマッピング）の典型的な例です。詳細は、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。現在のところ、Cisco Service Control Solution では IP アドレス、IP 範囲、VLAN のマッピングがサポートされています。

ブロッキング API とノンブロッキング API のいずれにも、パラメータとしてマッピングを受け入れる操作が含まれています。例は次の要素で構成されています。

- **addSubscriber** 操作（ブロッキング API）
- **login** メソッド（ブロッキング API またはノンブロッキング API）

API メソッドにマッピングを渡す場合、呼び出し側は次に示す 2 つのパラメータを要求されます。

- 文字列（**char***）マッピング識別子または文字列（**char****）マッピングの配列
- **MappingType** 列挙体または **MappingType** 変数の配列

配列を渡す場合は、**MappingType** 変数配列に、マッピング配列と同じ数の要素が含まれている必要があります。

API は、次に示すサブスクライバ マッピング型（**MappingType** 列挙体で定義）をサポートしています。

- IP アドレスまたは IP 範囲
- VLAN タグ

IP アドレス マッピングの指定

IP アドレスの文字列フォーマットには、一般的に次のような 10 進表記法が使用されています。

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

例：

- `216.109.118.66`

GeneralDefs.h ヘッダー ファイルには IP アドレスのマッピング型があります。

- **IP_RANGE** には、IP マッピング（マッピング識別子配列内の同じインデックスとマッピング識別子が一致する IP-Address または IP-Range）を指定します。

IP 範囲マッピングの指定

IP 範囲の文字列フォーマットは、10 進表記法の IP アドレスおよびビット マスク内の 1 の数を表す 10 進数です。

```
IP-Range=[0-255].[0-255].[0-255].[0-255]/[0-32]
```

例：

- **10.1.1.10/32** は、フル マスクの IP 範囲、つまり正規の IP アドレスです。
- **10.1.1.0/24** は、24 ビット マスクの IP 範囲で、**10.1.1.0** ～ **10.1.1.255** までの範囲のアドレスすべてを表します。

（注） IP 範囲のマッピングの型は、IP アドレスのマッピングの型と同じです。

VLAN タグ マッピングの指定

文字列は指定範囲の 10 進数です。

GeneralDefs.h ヘッダー ファイルにもマッピング型が含まれています。

VLAN は、マッピング識別子配列内の同じインデックスとマッピング識別子が一致する VLAN マッピングを指定します。

サブスクライバ ドメイン

ドメインとは、どの SCE デバイスをサブスクライバ レコードでアップデートしなければならないかを SM に伝える識別子です。ドメインの詳細については、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。

ドメイン名には型があります（**char***）。システム ドメイン名は、システム導入時にネットワーク管理者が決めるため、導入システムによってさまざまです。API には、サブスクライバが所属するドメインを指定したり、システムのドメイン名に関するクエリを許可したりするメソッドが含まれています。

ある API 操作で SM ドメイン リポジトリにないドメイン名が指定された場合、その操作はエラーとみなされ、**ERROR_CODE_DOMAIN_NOT_FOUND** エラーの **ReturnCode** が戻されます。

SM の自動ドメイン ローミング機能により、サブスクライバは最新のドメイン パラメータを指定して login メソッドを呼び出すことにより、各ドメイン間を移動できるようになります。

（注） 自動ドメイン ローミング機能は、旧バージョンの SM API とは互換性がありません。旧バージョンの SM API ではサブスクライバのドメインの変更はサポートされていないためです。

サブスクライバ プロパティ

サブスクライバ プロパティを扱う操作もいくつかあります。サブスクライバ プロパティは、サブスクライバによって生成されたネットワーク トラフィックに対する SCE の分析や反応に影響する一組のキー値です。

プロパティについて詳しくは、『*Cisco SCMS Subscriber Manager User Guide*』、および『*Cisco Service Control Application for Broadband (SCA BB) User Guide*』を参照してください。後者のマニュアルにはアプリケーション固有の情報が記載されています。システムで実行中のアプリケーションのサブスクライバ プロパティ、許可された値のセット、各プロパティ値の重要度が紹介されています。

C/C++ API 操作用のサブスクライバ プロパティをフォーマットするには、文字列配列（**char****）の **propertyKeys** および **propertyValues** を使用します。

（注） この配列は*同じ長さ*にする必要があり、また NULL エントリは許可されません。キー配列のキーごとに値配列に対応するエントリがあります。**propertyKeys[j]** の値は **propertyValues[j]** に格納されます。

例：

プロパティのキー配列が **{"packageId","monitor"}** で、値配列が **{"5","1"}** である場合、このプロパティは、**packageId=5, monitor=1** となります。

カスタム プロパティ

カスタム プロパティを扱う操作もあります。カスタム プロパティはサブスクライバ プロパティと似ていますが、サブスクライバのトラフィックに対する SCE の分析や操作には影響しません。アプリケーション管理モジュールはカスタム プロパティを使用して各サブスクライバに関する追加情報を保存します。

カスタム プロパティをフォーマットする場合は、サブスクライバ プロパティのフォーマットと同様に、文字列（**char****）配列の **customPropertyKeys** と **customPropertyValues** を使用します。

ロギング機能

API パッケージには Logger 抽象クラスがあります。このクラスは継承可能で、これを使用すると SM API をホスト アプリケーションのログに統合できます。**Logger** クラスは、基本的な 4 レベルのロギングを出力します。これらは、エラー メッセージ、警告メッセージ、情報メッセージ、数レベルからなるトレース メッセージです。この機能は、ブロッキングとノンブロッキングの両方の API にあります。**Logger.h** ヘッダー ファイルには **Logger** クラスがあります。

API ユーザは **Logger** クラスからの継承によってロガーを実装する必要があります。API がこのロガーを使用できるようにするには、コードで、C++ 実装 API の **setLogger** メソッドを呼び出さなければなりません。

テストや単純な LEG の実装用に、API パッケージには **PrintLogger** クラスがあります。このクラスは、ログ メッセージを標準エラー（STDERR）に出力する **Logger** クラスの単純実装です。API ユーザは **PrintLogger** オブジェクトを開始し、**PrintLogger** クラスの **setLoggingLevels** メソッドにより、ロギング レベルを設定したり、API の **setLogger** メソッドにより、API にこのロガー オブジェクトを渡したりすることができます。**PrintLogger.h** ヘッダー ファイルには **PrintLogger** クラスがあります。

切断コールバック リスナ

ブロッキングおよびノンブロッキング API には、API が SM から切断されたときに通知する切断コールバック リスナを設定できます。切断コールバック リスナは、次のように定義します。

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

切断リスナの設定には、**setDisconnectListener** メソッドを使用します。

切断コールバック リスナの例

次に、**stdout** にメッセージを出力して終了する切断コールバック リスナの単純実装の例を示します。

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

信号の処理

ネットワーク モジュールとしての SCMS SM C/C++ API は、SM がクローズするソケットを処理することがあります（「Broken Pipe」信号が生じる可能性のある SM の再起動時など）。UNIX 環境では、この信号を処理することを推奨します。

信号を無視する場合は、次の呼び出しを追加します。

```
sigignore(SIGPIPE);
```

使用時のヒント

コードを実装してアプリケーションに API を統合する場合は次のヒントを参考にしてください。

- SM に接続したら、API を何度も使用して API と SM との接続を維持します。接続は適切なタイミングで確立され、これにょって SM 側と API クライアント側にリソースが割り当てられます。
- スレッド間で API 接続を共有します。LEG ごとに接続を 1 つ確立することを推奨します。接続を複数にするには、SM とクライアント側にさらにリソースが必要になります。
- API への呼び出し同期を実行しないようにします。API への呼び出しはクライアントが自動的に同期させます。
- API クライアント（LEG）の配置順は、SM マシンのプロセッサの番号順にすることを推奨します。
- LEG アプリケーションがログオン操作時にバーストした場合は、内部バッファ サイズを拡張してバーストに対処します（ノンブロッキング方式）。
- 統合中は、SM ログに API 操作を出力するように SM **logon_logging_enabled** コンフィギュレーション パラメータを設定し、問題が発生した場合のトラブルシューティングに備えます。
- LEG アプリケーションをデバッグ モードで使用し、ノンブロッキング操作の戻り値を記録または出力します。
- SM への接続の復元力を向上させるには、自動再接続機能を使用します。
- クラスタのセットアップでは、API の接続にはクラスタの仮想 IP アドレスを使用し、任意のマシンの管理 IP アドレスは使用しないようにします。

CHAPTER 3

ブロッキング API

この章では、ブロッキング API の機能と動作について説明し、コードの例を紹介します。また、ブロッキング API 固有の機能である応答タイムアウトについても説明します。

(注) *自動統合*の開発だけが必要な場合は、この章は飛ばして第 4 章「ノンブロッキング API」へ進んでください。

マルチスレッドのサポート

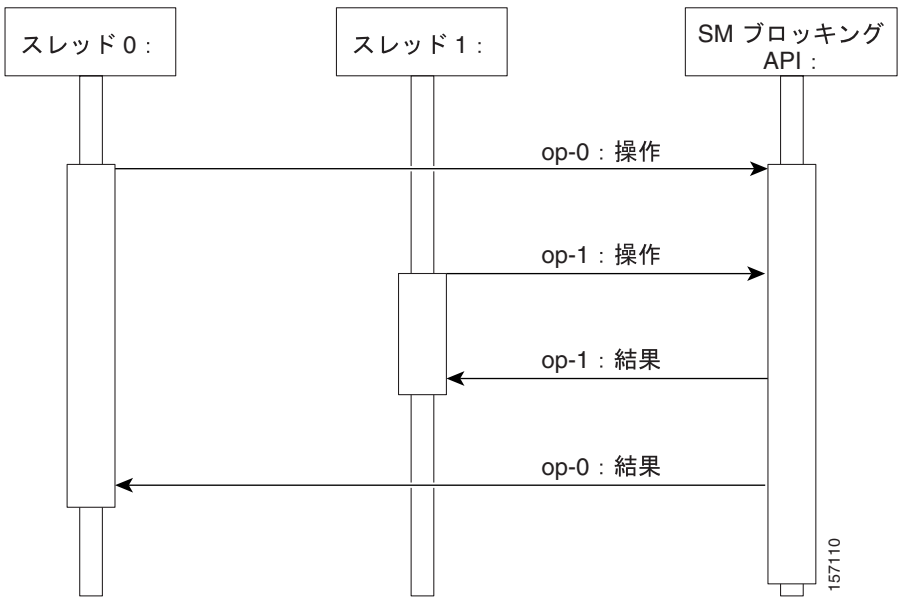
ブロッキング API では、メソッドを同時に呼び出すスレッドの数を設定できます。スレッド数の設定に関する詳細は、「C++ init メソッド」（p.3-32）を参照してください。

(注)　ブロッキング API でマルチスレッドにする場合、呼び出しの順序は保証**されません**。

マルチスレッドのサポートの例

スレッド -0 はタイム -0 で操作 -0 を呼び出し、スレッド -1 はタイム -1 で操作 -1 を呼び出します。このときタイム -1 はタイム -0 よりあとです。この例では、次の図（縦方向が時間）のように操作 -1 が操作 -0 よりも**前に**実行される可能性があります。

図 3-1　マルチスレッドのサポート



SM は 5 つのスレッドを割り当ててそれぞれの API インスタンスを処理します。複数のスレッドを持つ API を使用するマルチスレッド アプリケーションは、5 つのスレッド順に開発することを推奨します。より多くのスレッドを実装すると、呼び出しスレッドの遅延時間が長くなる可能性もあります。

操作タイムアウト エラー コード

ブロッキング操作は、SM から操作結果が取得された場合にだけ戻ります。ネットワークの誤作動やその他のエラーによって操作結果を取得できない場合、呼び出し側はいつまでも待機することになります。ただし SM API には、応答タイムアウト機能を使用して、こうした状況を避ける方法が用意されています。

呼び出し側は応答タイムアウト機能、すなわち **setReplyTimeout** メソッドを使用してタイムアウトを設定できます。タイムアウト期間内に応答が戻らない場合は、**ERROR_CODE_CLIENT_OPERATION_TIMEOUT** エラーで ReturnCode が戻ります。

応答タイムアウトを設定するには、**int** 値を指定して **setReplyTimeout** 機能を呼び出します。応答タイムアウトはミリ秒単位です。ゼロの値を指定すると、結果が届くまで操作が待機（フリーズまたは停止）状態になります。つまり、結果が届かなければ永久に待機状態のままになります。

ブロッキング API のメソッド

ここでは、ブロッキング API のメソッドについて説明します。メソッドの構文のあとに、各メソッド入力パラメータと戻り値の説明を示します。

ブロッキング API は、ノンブロッキング API の上位集合です。戻り値と結果の処理が異なる点を除けば、同じ操作の関数と構文の構造はどちらの API でも同じです。

C と C++ の API は、同じ関数シグニチャを共有していますが異なる点もあります。たとえばブロッキング C API では、関数名にはすべて **SMB_ プレフィクス**が付き、すべての関数の最初のパラメータは **SMB_HANDLE** 型の API ハンドルです。2 つの API のその他の相違点については、関数の説明で取り上げます。

ブロッキング API のサブスクライバ管理メソッドは次のカテゴリに分類されます。

- **IP およびプロパティの動的割り当て** — AAA システムとの統合に SM API を使用する場合は、次のようなメソッドを使用します。
  - — login（p.3-5）
  - — logoutByName（p.3-8）
  - — logoutByNameFromDomain（p.3-10）
  - — logoutByMapping（p.3-11）
  - — loginCable（p.3-13）
  - — logoutCable（p.3-15）

（注） これらのメソッドは、データベースに対するサブスクライバの追加や削除ではなく、既存のサブスクライバの動的パラメータ（IP アドレスなど）の変更に使用されます。

- **静的 / 手動のサブスクライバ設定** — たとえば GUI を利用する場合は、次のメソッドを使用します。
  - — addSubscriber（p.3-16）
  - — removeSubscriber（p.3-18）
  - — removeAllSubscribers（p.3-19）
  - — setPropertiesToDefault（p.3-29）
  - — removeCustomProperties（p.3-30）

- サブスクライバ認識モードで個別に実行する単純な読み取りのみの操作は、次のメソッドを使用します。
  - getNumberOfSubscribers（p.3-19）
  - getNumberOfSubscribersInDomain（p.3-20）
  - getSubscriber（p.3-21）
  - subscriberExists（p.3-22）
  - subscriberLoggedIn（p.3-23）
  - getSubscriberNameByMapping（p.3-24）
  - getSubscriberNames（p.3-25）
  - getSubscriberNamesInDomain（p.3-26）
  - getSubscriberNamesWithPrefix（p.3-27）
  - getSubscriberNamesWithSuffix（p.3-28）
  - getDomains（p.3-29）

異なるカテゴリのメソッドを 1 つのアプリケーションに組み合わせて使用できます。この分類は、説明の目的のみに示したものです。

- **API のメンテナンス、初期化、接続、切断に使用されるメソッド**
  - C++ setLogger メソッド（p.3-31）
  - C++ init メソッド（p.3-32）
  - C SMB_init 関数（p.3-33）
  - C SMB_release 関数（p.3-34）
  - setReconnectTimeout（p.3-34）
  - setName（p.3-35）
  - connect（p.3-35）
  - disconnect（p.3-36）
  - isConnected（p.3-36）

（注） メソッドの説明の最後に記載されている例は、C++ の例です。メソッドの最後に記載されている例にはそれぞれ、その前に次のようなサンプル コードが記されています。

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

- login（p.3-5）
- logoutByName（p.3-8）
- logoutByNameFromDomain（p.3-10）
- logoutByMapping（p.3-11）
- loginCable（p.3-13）
- logoutCable（p.3-15）
- addSubscriber（p.3-16）
- removeSubscriber（p.3-18）
- removeAllSubscribers（p.3-19）

login

構文

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

説明

login メソッドは、SM データベース内の既存のサブスクライバのドメイン、マッピング、プロパティの追加や変更を実行します。このメソッドは部分データで呼び出すことができます。たとえば、マッピングだけ、またはプロパティだけを指定したり、未変更フィールドに NULL を入力することが可能です。

同じドメイン内に同じ（つまり衝突する）マッピングを持つ別のサブスクライバがすでに存在している場合、既存のサブスクライバから衝突しているマッピングが削除され、新しいサブスクライバにそのマッピングが割り当てられます。

指定したサブスクライバが SM データベース内に存在しない場合は、与えられたデータでサブスクライバが作成されます。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argMappings ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。マッピングが指定されておらず **argIsAdditive** フラグが TRUE の場合、直前のマッピングが保持されます。直前のマッピングが存在しない場合、操作はエラーとなります。

argMappingTypes ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingsSize ― **argMappings** 配列、および **argMappingTypes** 配列のサイズです。

argPropertyKeys ― 「サブスクライバ プロパティ」（p.2-12）のプロパティのキーと値に関する説明を参照してください。

argPropertyValues ― 「サブスクライバ プロパティ」（p.2-12）のプロパティのキーと値に関する説明を参照してください。

argPropertySize ― **argPropertyKeys** 配列、および **argPropertyValues** 配列のサイズです。

argDomain ― 「サブスクライバ ドメイン」（p.2-11）を参照してください。

ドメインが NULL であっても、そのサブスクライバにすでにドメインがある場合は、既存のドメインが保持されます。

ドメインが、サブスクライバにすでに割り当てられているドメインとは異なる場合、既存のドメインの SCE からサブスクライバが自動的に削除され、新しいドメインの SCE に移されます。

ArgIsAdditive ― マッピング パラメータを参照してください。

- TRUE ― この呼び出しによって与えられたマッピングをサブスクライバ レコードに追加します。
- FALSE ― この呼び出しで与えられたマッピングで、サブスクライバ レコードの既存のマッピングを上書きします。

argAutoLogoutTime ― このメソッドに引数として与えられたマッピングのみに適用されます。

- 正の値（N）― N 秒後に自動的にマッピングからログアウトします（logout メソッドの呼び出しに似ています）。
- 0 の値 ― 指定されたマッピングのその時点での有効時間が維持されます。
- 負の値 ― 指定されたマッピングに設定されている有効時間を無効にします。

戻り値

エラーが発生していない場合は、void 型の **ReturnCode** 構造体へのポインタが戻ります。

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION
- ERROR_CODE_UNKNOWN

  このエラーは、次の場合に発生する可能性があります。

  - 存在しないサブスクライバまたはドメインのないサブスクライバの **domain** パラメータが NULL 値の場合
  - **propertyValues** パラメータの値が無効である場合

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

john という名前の既存のサブスクライバに IP アドレス 192.168.12.5 を追加し、マッピングは変更しない場合は、次のようにします。

```
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
```

IP アドレス 192.168.12.5 を追加して、直前のマッピングに上書きする場合は、次のようにします。

```
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
```

john に以前に割り当てられた 192.168.12.5 の自動ログアウト時間を延長するには、次のようにします。

```
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
```

john の動的プロパティ（package ID など）を変更するには、次のようにします。

```
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);
```

john という名前の既存のサブスクライバに IP アドレス 192.168.12.5 を追加し、既存のマッピングは変更せず、*john* の動的プロパティ（package ID など）を変更する場合は、次のようにします。

```
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

構文

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

説明

データベース内のサブスクライバを探し、そのサブスクライバからマッピングを削除します。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argMappings ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。マッピングが指定されていない場合は、すべてのサブスクライバ マッピングが削除されます。

argMappingTypes ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingsSize ― **argMappings** 配列、および **argMappingTypes** 配列のサイズです。

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― サブスクライバが見つかり、そのサブスクライバのマッピングがサブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでサブスクライバが見つからなかった場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- 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

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

サブスクライバ *john* の IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";
bapi.logoutByName(
"john",&ip_address,
&map_type,
1);
```

サブスクライバ *john* の IP アドレスをすべて削除するには、次のようにします。

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

logoutByNameFromDomain

構文

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

説明

指定されたドメインに基づいてデータベース内でサブスクライバを探し、そのサブスクライバからマッピングを削除します。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argMappings ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。マッピングが指定されていない場合は、すべてのサブスクライバ マッピングが削除されます。

argMappingTypes ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingsSize ― **argMappings** 配列、および **argMappingTypes** 配列のサイズです。

argDomain ― 「サブスクライバ ドメイン」（p.2-11）を参照してください。

次の条件の *いずれか* に該当する場合、操作はエラーとなります。

- ドメインがヌルであるが、そのサブスクライバがデータベース内に存在し、ドメインに所属している場合
- 指定されたドメインが間違っている場合

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― サブスクライバが見つかり、そのサブスクライバのマッピングがサブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでサブスクライバが見つからなかった場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- 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

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

ドメイン *subscribers* から、サブスクライバ *john* の IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
MappingType map_type = IP_RANGE;
char* ip_address = "192.168.12.5";
bapi.logoutByNameFromDomain(
"john",
&ip_address,
&map_type,
1,
"subscribers");
```

ドメイン *subscribers* から、サブスクライバ *john* の IP アドレスをすべてを削除するには、次のようにします。

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

logoutByMapping

構文

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

説明

ドメインおよびマッピングに基づくサブスクライバを探し、そのサブスクライバ マッピングを削除します。サブスクライバはデータベースに残ります。

パラメータ

argMapping ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingType ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argDomain ― logoutByNameFromDomain メソッドの「パラメータ」（p.3-10）の説明を参照してください。

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― サブスクライバが見つかり、サブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでサブスクライバが見つからなかった場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- 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

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

ドメイン **subscribers** から IP アドレス 192.168.12.5 を削除するには、次のようにします。

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

loginCable

構文

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

説明

ケーブル環境に適応した login メソッドで、SM 内でケーブル対応モジュールを呼び出します。このメソッドでは CPE が SM にログインされます。CM のログインには、CPE と CM の両方の引数に CM MAC アドレスを指定します。詳細は、『*Cisco SCMS Subscriber Manager User Guide*』の付録「Cable Environment」を参照してください。

（注） SM データベース内の CPE の名前は、CPE と CM の値を **2 つ**の下線 [“_”] 文字で連結したものとなります。呼び出し側は CPE と CM の値の合計の長さが **38** 文字を超えていないことを確認する必要があります。

パラメータ

argCpe ― CPE 固有の識別子（通常、MAC アドレス）

argCm ― ケーブル モデム固有の識別子（通常、MAC アドレス）

argIp ― CPE の IP アドレス

argLease ― CPE のリース時間

argDomain ― 「サブスクライバ ドメイン」（p.2-11）を参照してください。

通常は CMTS IP です。

（注） CMTS IP がドメイン名として正しく解釈されるためには、SM にドメインの別名を設定する必要があります。別名の設定については、『*Cisco SCMS Subscriber Manager User Guide*』の「Configuring Domains」を参照してください。

argPropertyKeys ― 「サブスクライバ プロパティ」（p.2-12）のキーと値に関する説明を参照してください。CPE のアプリケーション プロパティが一部またはまったく設定されていない場合、不足しているアプリケーション プロパティの値はこの CPE が属する CM のアプリケーション プロパティからコピーされます。CM の各アプリケーション プロパティは、その CM に属している CPE のデフォルト値として使用されます。

argPropertyValues ― 「サブスクライバ プロパティ」（p.2-12）のキーと値に関する説明を参照してください。

argPropertySize ― **argPropertyKeys** 配列、および **argPropertyValues** 配列のサイズです。

戻り値

void 型の **ReturnCode** 構造体へのポインタ

エラー コード

なし

例

CM1 という名前の CM に IP アドレス 192.168.12.5 を追加し、リース時間を 2 時間にするには、次のようにします。

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

CM1 に属する *CPE1* という名前の CPE に IP アドレス 192.168.12.50 を追加し、リース時間を 1 時間にするには、次のようにします。

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

logoutCable

構文

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

説明

SM ケーブル対応モジュールにログアウト イベント（CPE のオフライン）を示します。

パラメータ

argCpe ― loginCable メソッドの「パラメータ」（p.3-13）の説明を参照してください。

argCm ― loginCable メソッドの「パラメータ」（p.3-13）の説明を参照してください。

argIp ― loginCable メソッドの「パラメータ」（p.3-13）の説明を参照してください。

argDomain ― loginCable メソッドの「パラメータ」（p.3-13）の説明を参照してください。

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― その CPE が見つかり、サブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでその CPE が見つからなかった場合

エラー コード

なし

例

CM1 に属する *CPE1* から IP アドレス 192.168.12.5 を削除するには、次のようにします。

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

addSubscriber

構文

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

説明

与えられたデータに基づいて新しいサブスクライバ レコードを 1 つ作成し、これを SM データベースに追加します。この名前のサブスクライバがすでに存在する場合、そのサブスクライバは新しいサブスクライバが追加される前に削除されます。**login** では値が渡されたフィールドが変更され、未指定フィールドは変更されませんが、**addSubscriber** では、値が渡されたパラメータによって、サブスクライバが指定どおりに設定されます。

（注） 既存のサブスクライバには、**addSubscriber** ではなく **login** メソッドを呼び出すことを推奨します。動的なマッピングおよびプロパティは **login** を使用して設定する必要があります。静的なマッピングおよびプロパティは、**addSubscriber** を使用してサブスクライバの初回作成時に設定する必要があります。

（注） **addSubscriber** を使用する場合、自動ログアウト機能は常にディセーブルです。自動ログアウトをイネーブルにするには、**login** を指定します。

例：

サブスクライバ データベース内にすでにセットアップされているサブスクライバ *AB* には、*IP1* という IP マッピングが 1 つあります。

AB に対する **addSubscriber** 操作がマッピングの指定なしに呼び出されると（**mappings** と **mappingTypes** のフィールドが NULL）、*AB* はマッピングがないままになります。

ただし、これらの NULL 値パラメータを使って **login** 操作を呼び出しても *AB* のマッピングは変更されず、*AB* は直前の IP マッピング *IP1* を保持したままになります。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argMappings ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingTypes ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingsSize ― **argMappings** 配列、および **argMappingTypes** 配列のサイズです。

argPropertyKeys ― 「サブスクライバ プロパティ」（p.2-12）のプロパティのキーと値に関する説明を参照してください。

argPropertyValues ― 「サブスクライバ プロパティ」（p.2-12）のプロパティのキーと値に関する説明を参照してください。

argPropertySize ― **argPropertyKeys** 配列、および **argPropertyValues** 配列のサイズです。

argCustomPropertyKeys ― 「カスタム プロパティ」（p.2-12）のカスタム プロパティのキーと値に関する説明を参照してください。

argCustomPropertyValues ― 「カスタム プロパティ」（p.2-12）のカスタム プロパティのキーと値に関する説明を参照してください。

argPropertySize ― **argCustomPropertyKeys** 配列、および **argCustomPropertyValues** 配列のサイズです。

argDomain ― 「サブスクライバ ドメイン」（p.2-11）を参照してください。

NULL 値は、サブスクライバがドメインレスであることを示します。

戻り値

void 型の **ReturnCode** 構造体へのポインタ

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- 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

  このエラー コードは、**propertyValues** パラメータに無効な値が指定されたことを示します。

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

カスタム プロパティを指定して新しいサブスクライバ *john* を追加するには、次のようにします。

```
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

構文

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

説明

SM データベースから 1 つのサブスクライバを完全に削除します。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― データベースでサブスクライバが見つかり、正常に削除された場合
- FALSE ― TRUE 条件に合致しない場合。つまり、データベースでサブスクライバが見つからなかったか、または見つかったが正常に削除されなかった場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

サブスクライバ *john* をデータベースから完全に削除するには、次のようにします。

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

removeAllSubscribers

構文

```
ReturnCode* removeAllSubscribers()
```

説明

SM からすべてのサブスクライバを削除し、データベースにサブスクライバが 1 つもない状態にします。

（注） このメソッドは実行に時間がかかる場合もあります。操作タイムアウトによる除外が発生しないようにするため、このメソッドを呼び出す前に操作タイムアウトを大きな値（最大 5 分）に設定してください。

戻り値

void 型の **ReturnCode** 構造体へのポインタ

エラー コード

なし

getNumberOfSubscribers

構文

```
ReturnCode* getNumberOfSubscribers()
```

説明

SM データベース内のサブスクライバの合計数を取得します。

戻り値

SM 内のサブスクライバ数を表す整数が格納された **ReturnCode** 構造体へのポインタ

エラー コード

なし

getNumberOfSubscribersInDomain

構文

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

説明

ある 1 つのサブスクライバ ドメイン内のサブスクライバ数を取得します。

パラメータ

argDomain ― SM のドメイン リポジトリにあるサブスクライバ ドメインの名前

戻り値

所定のドメイン内のサブスクライバ数を表す整数が格納された **ReturnCode** 構造体へのポインタ

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE _DOMAIN_NOT_FOUND

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriber

構文

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

説明

サブスクライバ レコードを取得します。各フィールドは、整数、文字列、または文字列配列としてフォーマットされます。このメソッドの戻り値に関する説明を参照してください。SM データベース内にそのサブスクライバが存在しない場合は、例外処理が実行されます。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

戻り値

9 つの要素を持つ **ReturnCode** 構造体の配列が格納された **ReturnCode** 構造体へのポインタ。NULL 値となる配列要素はありません。

要素の値と意味は次のとおりです。

インデックス 0	サブスクライバ名（**char***）
インデックス 1	マッピングの配列（**char****）
インデックス 2	マッピングの型の配列（**short***）
インデックス 3	ドメイン名（**char***）
インデックス 4	プロパティ名の配列（**char****）
インデックス 5	プロパティ値の配列（**char****）
インデックス 6	プロパティ名の配列（**char****）
インデックス 7	カスタム プロパティ値の配列（**char****）
インデックス 8	自動ログアウト時間（現在からの秒数）の配列、またはマッピング（インデックス 1）ごとに設定されていない場合（long 1*）は -1

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

john のサブスクライバ レコードを取得するには、次のようにします。

```
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

構文

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

説明

あるサブスクライバが SM データベース内に存在していることを確認します。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

戻り値

ブール型の ReturnCode 構造体へのポインタ

- TRUE ― SM データベースでそのサブスクライバが見つかった場合
- FALSE ― そのサブスクライバが見つからなかった場合

エラー コード

なし

subscriberLoggedIn

構文

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

説明

SM データベース内の既存のサブスクライバがログインしているかどうかを確認します。つまり、そのサブスクライバが SCE データベースにも存在しているかどうか確認します。

SM が *Pull モード*で機能するように設定されている場合、このメソッドで TRUE の値が戻っても、そのサブスクライバが実際に SCE データベース内に存在しているとは**限りません**。単に必要に応じて SCE がサブスクライバを pull できるようになっていることを意味します。

SM データベース内にそのサブスクライバが存在しない場合は、例外がスローされます。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

戻り値

ブール型の **ReturnCode** 構造体へのポインタ

- TRUE ― そのサブスクライバがログインしている場合
- FALSE ― そのサブスクライバがログインしていない場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNameByMapping

構文

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

説明

マッピングおよびドメインに基づいてサブスクライバ名を検索します。

パラメータ

argMapping ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argMappingType ― 「ネットワーク ID マッピングについて」（p.2-10）のマッピングおよびマッピングの型についての説明を参照してください。

argDomain ― そのサブスクライバが属しているドメインの名前。次の条件の*いずれか*に該当する場合、操作はエラーとなります。

- ドメインがヌルであるが、そのサブスクライバがデータベース内に存在し、ドメインに所属している場合
- 指定されたドメインが間違っている場合

戻り値

文字列（**char***）型の **ReturnCode** 構造体へのポインタ

- サブスクライバ名 ― サブスクライバ レコードが見つかった場合
- NULL ― 与えられたマッピングを持つサブスクライバ レコードが SM データベース内で見つからなかった場合

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE _BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNames

構文

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

説明

SM データベースから、サブスクライバ名を一括して取得します。**argFirstName** から始まり、アルファベット順に **argAmount** の数だけサブスクライバが取得されます。

argFirstName が NULL の場合、SM データベース内に存在するサブスクライバ名のうちアルファベット順で最初のサブスクライバ名が使用されます。

（注） **getNumOfSubscribers** からの戻り値が必ずサブスクライバの合計数（全バルク）であるとは**限りません**。たとえば、取得中にいくつかのサブスクライバの追加や削除が実行された場合は合計数と異なる可能性があります。

パラメータ

argFirstName ― 最後のバルクの最後のサブスクライバ名（検索する最初の名前）。アルファベット順で最初のサブスクライバから開始する場合は、NULL を使用します。

argAmount ― 取得するサブスクライバの数を限定します。この値が SM の制限値（1000）より大きい場合は、SM の制限値が使用されます。

（注） このパラメータに 500 を超える値を指定することは**推奨しません**。

戻り値

アルファベット順のサブスクライバ名リストが格納された文字列配列型（**char****）の **ReturnCode** 構造体へのポインタ。

このメソッドは、要求されたサブスクライバから開始して、SM データベース内で見つかったすべてのサブスクライバを戻します。配列のサイズは、**argAmount** の最小値と SM 制限値（1000）によって制限されます。

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

例

サブスクライバ名をアルファベット順のリストで取得するには、次のようにします。

```
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

構文

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

説明

指定されたドメインに関連付けられているサブスクライバを SM データベースから取得します。

この操作の関数は、「getSubscriberNames」（p.3-25）操作の場合と同じです。

パラメータ

argFirstName ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argAmount ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argDomain ― SM のドメイン リポジトリにあるサブスクライバ ドメインの名前

戻り値

指定されたドメインに属しているサブスクライバ名のアルファベット順配列。

詳細については、getSusbcriberNames 操作の「戻り値」（p.3-25）の説明を参照してください。

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _DOMAIN_NOT_FOUND
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNamesWithPrefix

構文

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

説明

指定されたプレフィクスから名前が始まるサブスクライバを SM データベースから取得します。

この操作の関数は、「getSubscriberNames」（p.3-25）操作の場合と同じです。

パラメータ

argFirstName ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argAmount ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argPrefix ― 要求対象のサブスクライバ名のプレフィックスを示す文字列（大文字と小文字が区別されます）。

戻り値

要求されたプレフィクスから始まるサブスクライバ名のアルファベット順配列。

詳細については、getSusbcriberNames 操作の「戻り値」（p.3-25）の説明を参照してください。

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNamesWithSuffix

構文

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

説明

指定されたサフィックスが名前の最後に付いているサブスクライバを SM データベースから取得します。

この操作の関数は、「getSubscriberNames」（p.3-25）操作の場合と同じです。

パラメータ

argFirstName ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argAmount ― getSusbcriberNames 操作の「パラメータ」（p.3-25）の説明を参照してください。

argSuffix ― 要求対象のサブスクライバ名のサフィックスを示す文字列（大文字と小文字が区別されます）。

戻り値

要求されたサフィックスで終わるサブスクライバ名のアルファベット順配列。

詳細については、getSusbcriberNames 操作の「戻り値」（p.3-25）の説明を参照してください。

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

getDomains

構文

```
ReturnCode* getDomains()
```

説明

SM ドメイン リポジトリ内の現在のサブスクライバ ドメインのリストを提供します。

戻り値

SM 内のサブスクライバ ドメイン名の完全リストが格納された文字列配列型（**char****）の **ReturnCode** 構造体へのポインタ

エラー コード

なし

setPropertiesToDefault

構文

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

説明

1 つのサブスクライバの指定されたアプリケーション プロパティをリセットします。アプリケーションがインストールされている場合、該当するアプリケーション プロパティは、その時点でロードされているアプリケーション情報に基づいて、そのプロパティのデフォルト値に設定されます。アプリケーションがインストールされていない場合は、エラー コード **ERROR_CODE_ILLEGAL_STATE** が戻ります。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argPropertyKeys ― 「サブスクライバ プロパティ」（p.2-12）のプロパティのキーと値に関する説明を参照してください。

argPropertySize ― **argPropertyKeys** 配列のサイズです。

戻り値

void 型の **ReturnCode** 構造体へのポインタ

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- 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

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

removeCustomProperties

構文

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

説明

1 つのサブスクライバの指定されたカスタム プロパティをリセットします。

パラメータ

argName ― 「サブスクライバ名のフォーマット」（p.2-9）を参照してください。

argCustomPropertyKeys ― 「カスタム プロパティ」（p.2-12）のカスタム プロパティのキーと値に関する説明を参照してください。

argCustomPropertySize ― **argCustomPropertyKeys** 配列のサイズです。

戻り値

void 型の **ReturnCode** 構造体へのポインタ

エラー コード

このメソッドが戻す可能性があるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE _SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの詳細については、「エラー コードのリスト」（p.A-1）を参照してください。

C++ setLogger メソッド

構文

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

説明

Logger 抽象クラスの実装を設定します。このメソッドは、SM API ログ メッセージをホスト アプリケーション ログに統合するために使用します。

パラメータ

argLogger ― Logger 抽象クラスの実装です。

戻り値

なし

C++ init メソッド

構文

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

説明

API を設定し初期化します。

(注)　このメソッドは、C++ API の操作を実行する前に呼び出す必要があります。

パラメータ

argSupportedThreads ― API がサポートするスレッド数

argThreadPriority ― PRPC プロトコル ネットワーク スレッドのプライオリティ

argBufferSize ― 内部バッファ サイズ（デフォルト値は 2,000,000 バイト）

argKeepAliveDuration ― PRPC プロトコルのキープアライブ メッセージ間の必要な遅延に関するヒント（デフォルト値は 10 秒）

argConnectionTimeout ― 応答のない PRPC プロトコル接続の必要なタイムアウトに関するヒント（デフォルト値は 20 秒）

argReconnectTimeout ― SM への接続がダウンしたとき API が接続を再開しようとするまでのタイムアウト時間

戻り値

ブール値

- TRUE ― 成功
- FALSE ― 失敗

例

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

C SMB_init 関数

構文

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

説明

API の割り当て、設定、初期化を行います。

（注） このメソッドは、C API の操作を実行する前に呼び出す必要があります。

パラメータ

argSupportedThreads ― API がサポートするスレッド数

argThreadPriority ― PRPC プロトコル ネットワーク スレッドのプライオリティ

argBufferSize ― 内部バッファ サイズ（デフォルト値は 2,000,000 バイト）

argKeepAliveDuration ― PRPC プロトコルのキープアライブ メッセージ間の必要な遅延に関するヒント（デフォルト値は 10 秒）

argConnectionTimeout ― 応答のない PRPC プロトコル接続の必要なタイムアウトに関するヒント（デフォルト値は 20 秒）

戻り値

API への **SMB_HANDLE** ハンドル。このハンドルが NULL の場合、初期化は失敗しました。失敗していない場合は NULL 以外の値が戻ります。

例

```
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 関数

構文

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

説明

API で使用されたリソースを解放します。API を使用した場合、最後にこの関数を呼び出す必要があります。

パラメータ

argApiHandle ― **SMB_init** 関数を使用して受け取った API ハンドル

戻り値

なし

setReconnectTimeout

構文

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

説明

再接続タイムアウトを設定します。

SM への接続がダウンしたとき API は「再接続タイムアウト」の秒数が過ぎると接続を再確立しようとします。

パラメータ

reconnectTimeout ― タイムアウト

戻り値

なし

setName

構文

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

説明

API の名前を設定します。この名前はその API-SM 接続固有の識別子として使用されます。connect メソッドを呼び出す前に、setName 関数を呼び出す必要があります。

パラメータ

argName ― API 名

戻り値

なし

connect

構文

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

説明

SM への PRPC プロトコル接続の確立を試行します。

パラメータ

argHostName ― SM の IP アドレスまたはホスト名

argPort ― SM に接続する TCP ポート（デフォルトは 14374）

戻り値

ブール値

- TRUE ― 成功
- FALSE ― 失敗

disconnect

構文

```
bool disconnect()
```

説明

SM への PRPC プロトコル接続の終了を試行します。

戻り値

ブール値

- TRUE ― 成功
- FALSE ― 失敗

isConnected

構文

```
bool isConnected();
```

説明

SM への PRPC プロトコル接続が確立され稼働中であるかどうかを確認します。

戻り値

ブール値

- TRUE ― 接続が確立されています。
- FALSE ― 接続は確立されていません。

ブロッキング API C++ コードの例

ここでは、次の 2 つのコード例を紹介します。

サブスクライバ数の取得

次の例では、SM データベース内のサブスクライバ総数と各サブスクライバ ドメイン内のサブスクライバ数を **stdout** に出力します。

```
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;
}
```

サブスクライバの追加、情報の出力、サブスクライバの削除

次に、サブスクライバ データベースにサブスクライバを追加し、その情報を取得して stdout に出力し、サブスクライバをサブスクライバ データベースから削除するプログラムを示します。

```
#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);
}
}
```

ブロッキング API C コードの例

ここでは、次の 2 つのコード例を紹介します。

サブスクライバ数の取得

次の例では、SM データベース内のサブスクライバ総数と各サブスクライバ ドメイン内のサブスクライバ数を stdout に出力します。

```
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;
}
```

サブスクライバの追加、情報の出力、サブスクライバの削除

次に、サブスクライバ データベースにサブスクライバを追加し、その情報を取得して **stdout** に出力し、サブスクライバをサブスクライバ データベースから削除するプログラムを示します。

```
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

ノンブロッキング API

この章では、ノンブロッキング API の機能と動作について説明し、コードの例を紹介します。

また、ノンブロッキング API 固有の機能である結果ハンドラ コールバックについても説明します。

マルチスレッドのサポート

ノンブロッキング API では、メソッドを同時に呼び出すスレッドの数に制限がありません。

（注） ノンブロッキング API でマルチスレッドにする場合、呼び出しの順序が**保証されます**。API は、呼び出された順番で操作を実行します。

結果ハンドラ コールバック

ノンブロッキング API により、結果ハンドラ コールバックを設定できます。結果ハンドラ コールバックは、**handleSuccess** と **handleError** 用の 2 つの関数です（以下のコードを参照）。

```
/* operation failure callback specification */
typedef void (*OperationFailCallBackFunc)(Uint32 argHandle,
ReturnCode *argReturnCode);
/* operation success callback specification */
typedef void (*OperationSuccessCallBackFunc)(Uint32 argHandle,
ReturnCode *argReturnCode);
```

API を通じて実行した操作結果の成功またはエラーについて通知を受けたい場合は、これらのコールバックを実装する必要があります。

（注） 結果ハンドラ コールバックは、結果を取得するための**唯一**のインターフェイスです。API メソッドが呼び出し側に戻った直後に結果を戻すことは**できません**。

handleSuccess と **handleError** のコールバックは、両方とも次に示す 2 つのパラメータを受け入れます。

- **Handle** ― 各 API 操作の結果値は、**Uint32** 型のハンドルです。このハンドルによって、操作の呼び出しとその結果を関連付けることができます。ハンドル値 **X** を指定して **handle...** 操作が呼び出された場合、その結果は同じハンドル値（**X**）を呼び出し側に返した操作と一致します。
- **Result** ― **ResultCode** 型のポインタとして戻った実際の操作結果
- 結果ハンドラ コールバックの例（p.4-2）

結果ハンドラ コールバックの例

次の例は、成功またはエラーの操作数をカウントする結果ハンドラの簡単な実装です。この main メソッドは API を開始し、結果ハンドラを割り当てます。

結果ハンドラを正しく機能させるためには、以下の例に示されているコード シーケンスを守る必要があります。

（注） この例は、コールバック ハンドルの使用方法を示すものでは**ありません**。

```
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;
}
```

ノンブロッキング API のメソッド

ここでは、ノンブロッキング API のメソッドについて説明します。

メソッドによっては、負以外の **int** 型ハンドルを戻すものがあります。このハンドラにより操作の呼び出しとその結果を関連付けることができます（詳細については、「結果ハンドラ コールバック」[p.4-2] を参照してください）。内部エラーが発生すると、負の値が戻り、操作は実行されません。

結果ハンドラ コールバックに渡される操作は、「ブロッキング API」（p.3-1）の同じメソッドで説明した戻り値と同じですが、**Void** の戻り値が **NULL** に変換される点が異なります。

（注） エラーによってエラー コールバックが渡されるのは、ブロッキング API の一致する操作が、呼び出し時の SM データベースの状態に応じて、同じ引数を持つエラー コードを返す場合**だけ**です。

C と C++ API は同じ関数シグニチャを共有しますが、ノンブロッキング C API の関数名にはすべて **SMNB_** プレフィクスが付き、すべての関数の最初のパラメータは **SMNB_HANDLE** 型の API ハンドルである点が異なります。2 つの API のその他の相違点については、関数の説明で取り上げます。

この章で説明するメソッドは次のとおりです。

login

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「login」（p.3-5）の操作を参照してください。

logoutByName

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「logoutByName」（p.3-8）の操作を参照してください。

logoutByNameFromDomain

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「logoutByNameFromDomain」（p.3-10）の操作を参照してください。

logoutByMapping

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「logoutByMapping」（p.3-11）の操作を参照してください。

loginCable

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、ブロッキング API の章の「loginCable」（p.3-13）の操作を参照してください。

logoutCable

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「logoutCable」（p.3-15）の操作を参照してください。

C++ setLogger メソッド

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「C++ setLogger メソッド」（p.3-31）の操作を参照してください。

C++ init メソッド

構文

```
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)
```

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「C++ init メソッド」（p.3-32）の操作を参照してください。

C SMNB_init 関数

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「C SMB_init 関数」（p.3-33）の操作を参照してください。

戻り値

API への **SMNB_HANDLE** ハンドル。このハンドルが NULL の場合、初期化は失敗しました。失敗していない場合は NULL 以外の値が戻ります。

C SMNB_release 関数

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「C SMB_release 関数」（p.3-34）の操作を参照してください。

setReconnectTimeout

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「setReconnectTimeout」（p.3-34）の操作を参照してください。

setName

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「setName」（p.3-35）の操作を参照してください。

connect

構文

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

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「connect」（p.3-35）の操作を参照してください。

disconnect

構文

```
bool disconnect()
```

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「disconnect」（p.3-36）の操作を参照してください。

isConnected

構文

```
bool isConnected()
```

この操作の関数は、対応するブロッキング API 操作の関数と同じです。詳細については、「isConnected」（p.3-36）の操作を参照してください。

ノンブロッキング API C++ コードの例

ここでは、サブスクライバのログインおよびログアウトのコード例を紹介します。

ログインおよびログアウト

次の例では、事前定義された数のサブスクライバが SM にログインしたのち、ログアウトします。*切断リスナ*と*結果ハンドラ*が実装されている点に注意してください。

```
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();
}
```

ノンブロッキング API C コードの例

ここでは、サブスクライバのログインおよびログアウトのコード例を紹介します。

ログインおよびログアウト

次の例では、事前定義された数のサブスクライバが SM にログインしたのち、ログアウトします。*切断リスナ*と*結果ハンドラ*が実装されている点に注意してください。

```
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

エラー コードのリスト

この章では、C/C++ API で使用されるエラー コードのリストを紹介します。

エラー コードのリスト

エラー コードは、実際にどのようなエラーが原因で **ReturnCode**（**ErrorCode** が格納されている）が戻されたのかを知るために役立ちます。

エラー コード列挙体は、**GeneralDefs.h** ヘッダー ファイルに指定されています。次にエラー コードのリストとコードの説明を示します。

表 A-1　エラー コードのリスト

エラー コード	説明
ERROR_CODE_BAD_SUBSCRIBER_MAPPING	マッピングのフォーマット不良またはサブスクライバへのマッピングの不正割り当てです。
ERROR_CODE_DOMAIN_NOT_FOUND	操作で指定されたドメインは SM ドメイン リポジトリに存在していません。
ERROR_CODE_ILLEGAL_ARGUMENT	メソッドに指定された引数の 1 つが無効です。
ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME	指定されたサブスクライバ名が 40 文字を超えているか、または使用できない文字が含まれています。
ERROR_CODE_NOT_A_SUSBCRIBER_DOMAIN	操作で指定されたドメインは SM ドメイン リポジトリに存在していますが、サブスクライバ ドメインではありません。
ERROR_CODE_NUMBER_FORMAT	その API に指定された VLAN マッピング文字列は 10 進数ではありません。
ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST	操作の実行対象であるサブスクライバは、SM データベース内に存在していません。
ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION	サブスクライバは SM データベースに存在していますが、操作で指定されたドメインとは異なるドメインに関連付けられています。
ERROR_CODE_SUBSCRIBER_MAPPING_CONGESTION	操作でそのサブスクライバに指定されたマッピングは、すでにほかのサブスクライバに属しています。

表 A-1　エラー コードのリスト（続き）

エラー コード	説明
ERROR_CODE_SUSBSCRIBER_ALREADY_EXISTS	操作の実行対象となったサブスクライバは、すでに SM データベース内に存在しています。
ERROR_CODE_DATABASE_EXCEPTION	SM の内部エラー — 操作中にデータベース エラーが発生しました。
ERROR_CODE_ARRAY_ACCESS	SM の内部エラー
ERROR_CODE_ATTRIBUTE_NOT_FOUND	SM の内部エラー
ERROR_CODE_CLASS_CAST	SM の内部エラー
ERROR_CODE_CLASS_NOT_FOUND	SM の内部エラー
ERROR_CODE_CLIENT_INTERNAL_ERROR	内部エラー
ERROR_CODE_CLIENT_OUT_OF_THREADS	内部エラー
ERROR_CODE_ILLEGAL_STATE	SM の内部エラー
ERROR_CODE_OBJECT_NOT_FOUND	SM の内部エラー
ERROR_CODE_OPERATION_NOT_FOUND	SM の内部エラー
ERROR_CODE_OUT_OF_MEMORY	SM の内部エラー
ERROR_CODE_RUNTIME	SM の内部エラー
ERROR_CODE_NULL_POINTER	SM の内部エラー
ERROR_CODE_SE_ERROR	SM の内部エラー。SM はその SCE デバイスに対して操作を実行できませんでした。
ERROR_CODE_UNKNOWN	SM または API の内部エラー
ERROR_CODE_CLIENT_OPERATION_TIMEOUT	ブロッキング API 操作の結果が応答タイムアウト期限内に戻りませんでした。