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、これによって 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。