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);
}
```