User's
Manual

DL850E/DL850EV フリーラン Application Programming Interface ユーザーズマニュアル

YOKOGAWA
横河メータ&インスツルメンツ株式会社

IM B8074XW-01JA
初版

はじめに

このユーザーズマニュアルは、DL850E/DL850EV シリーズ向けフリーラン API（ScAPI.dll）の取り扱い上の注意 / 機能 /API 仕様などについて説明したものです。
ご使用中にこのマニュアルをよくお読みいただき、正しくお使いください。お読みになったあとは、ご使用時にすぐにご覧になれるところに、大切に保存してください。ご使用中に操作がわからなくなったときなどにきっとお役にたちます。
なお、DL850E/DL850EV シリーズの取り扱い上の注意 / 機能 / 操作方法、Windows の取り扱い / 操作方法などについては、それぞれのマニュアルをご覧ください。

ご注意

- 本書の内容は、性能 / 機能の向上などにより、将来予告なしに変更することがあります。また、実際の画面表示内容が、本書に記載の画面表示内容と多少異なることがあります。
- 本書の内容に関しては万全を期しておりますが、万一ご不審の点や誤りなどお気づきのことがありましたら、お手数ですが、当社支社 / 支店 / 営業所までご連絡ください。

商標

- Windows 7、Windows 8、Windows 8.1 および Windows 10 は、米国 Microsoft Corporation の、米国およびその他の国における登録商標または商標です。
- 本文中の各社の登録商業または商標には、®、TM マークは表示していません。
- その他、本文中に使われている会社名 / 商品名は、各社の登録商標または商標です。

履歴

2016 年 3 月 初版発行

1st Edition: March 2016(YMI)

目次

第 1 章　ソフトウエア概要

1.1　ソフトウエア概要 1-1

第 2 章　ご使用にあたっての注意

2.1　ご使用にあたっての注意 2-1

第 3 章　フリーラン API 概要

3.1　フリーラン API 概要 3-1
3.2　API 機能概要 3-2
3.3　API 利用の流れ 3-3

第 4 章　API 機能仕様

4.1　クラス定義 4-1
4.2　固定値定義 4-2
4.3　API 詳細仕様 4-3
4.4　DLL リンク方式と配置 4-18

1.1　ソフトウエア概要

概要

本ソフトウエア (ScAPI.dll) は、DL850E/DL850EV シリーズのフリーランモードにおける、波形データ取得のための API(Application Program Interface) を提供するものです。

機能

本ソフトウエアを利用して、以下の機能を実現することができます。詳細機能につきましては、API 詳細仕様をご参考ください。

- API の初期化
- 測定器への接続と切断
- 各種パラメータ設定
- 波形データの取得

ソフトウエア構成

本ソフトウエアは以下のパッケージ構成からなります。

- フリーラン API ユーザーズマニュアル (本書)
- API 利用関連ファイル (下表)

ファイル名	内容
ScAPI.dll	フリーラン API 本体
ScAPI64.dll	フリーラン API 本体 64bit 版
ScAPI.lib	フリーラン API インポートライブラリ （C++ のみ）
ScAPI.h	関数プロトタイプ宣言のヘッダファイル（C++ のみ）
ScAPINet.dll	.NET 版フリーラン API ライブラリ
tmctl.dll	通信用ライブラリ
tmctl64.dll	通信用ライブラリ 64bit 版
YKMUSB.dll	USB 通信用ライブラリ
YKMUSB64.dll	USB 通信用ライブラリ 64bit 版

システム条件

- パーソナルコンピュータ本体
  以下の条件を有したパーソナルコンピュータの動作環境が必要です。
  OS(オペレーティングシステム):
  　日本語、または英語 Microsoft Windows 7(SP1 以降)、Windows8, Windows8.1 または Windows 10
  CPU: Core2Duo 2GHz 以上
  メモリ : 1GB 以上 (2GB 以上を推奨)
- 開発環境
  Visual Studio 2008 以降 .NET Framework 3.5 以降

2.1　ご使用にあたっての注意

免責

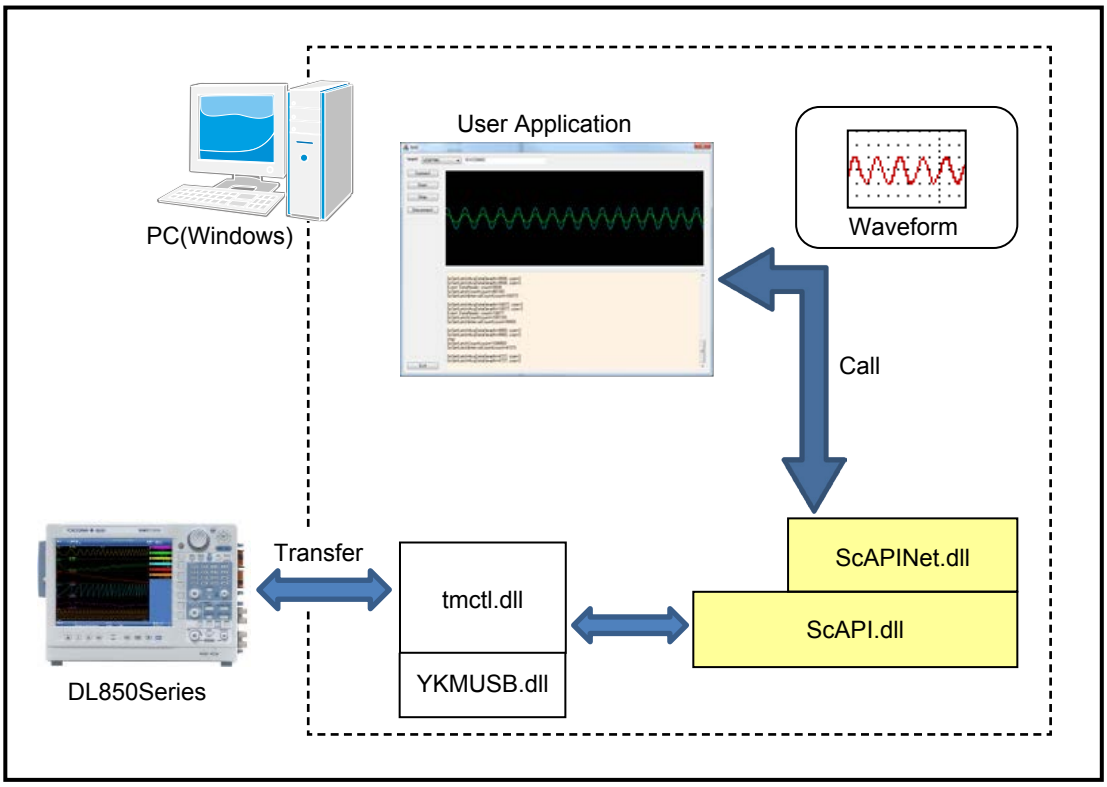
本ソフトウエアの使用に関して、直接または間接に生じるいっさいの損害について、責任を負いません。

使用上の注意

- 本ソフトウエアは、DL850E/DL850EV シリーズのフリーランモード専用ライブラリです。他の製品には使用できません。
- 本ソフトウエアのバージョン、および使用する DL850E/DL850EV のファームウエアバージョンをご確認の上、ご使用ください。

3.1　フリーラン API 概要

API は、ダイナミックリンクライブラリ (DLL:Dynamic Link Library) として提供されます。ユーザー作成のアプリケーションと一緒に本 DLL をリンクすることで、API を利用できます。
フリーラン API は下図のように、フリーラン動作中の波形データの収集および測定条件の設定に関する機能をアプリケーションに提供します。



3.2 API 機能概要

API の機能概要について説明します。

初期化 / 終了

API の初期化・終了に関する API を以下に示します。

API Name	Function	Page
ScInit	API の初期化（使用開始）	4-3
ScExit	API の使用終了	4-3

接続 / 切断

測定器への接続および切断に関する API を以下に示します。

API Name	Function	Page
ScOpenInstrument	測定器に接続し、接続ハンドルを取得する	4-4
ScCloseInstrument	測定器から切断する	4-4

測定条件の設定と取得

測定条件の設定および取得に関する API を以下に示します。

API Name	Function	Page
ScSetControl	通信コマンドを測定器に送信する	4-5
ScGetControl	コマンド応答を測定器から受信する	4-5
ScQueryMessage	通信コマンドを送信し応答を受信する	4-7
ScGetBinaryData	バイナリデータを受信する	4-6
ScSetSamplingRate	サンプリングレートを設定する	4-12
ScGetSamplingRate	サンプリングレートを取得する	4-12
ScGetBaseSamplingRate	基本サンプリングレートを取得する	4-12
ScGetChannelSamplingRatio	基本サンプリングレートとの比率を取得する	4-13
ScStart	測定を開始する	4-8
ScStop	測定を停止する	4-8

フリーラン情報の取得

フリーラン情報の取得に関する API を以下に示します。

API Name	Function	Page
ScGetLatchCount	ラッチ位置を取得する	4-9
ScGetLatchIntervalCount	前回ラッチした位置から今回ラッチした位置までのサンプル数を取得する	4-9
ScGetChannelDelay	チャネルの位相差点数を取得する	4-11
ScGetStartTime	測定開始時刻を取得する	4-11
ScChannelBits	該当チャネル 1 データあたりのビット数を取得する	4-13
ScGetChannelGain	チャネルのゲイン値を取得する（波形データを実データ値に変換する場合に使用する）	4-14
ScGetChannelOffset	チャネルのオフセット値を取得する（波形データを実データ値に変換する場合に使用する）	4-14
ScSetDataReadyCount	データレディイベントを発生させる点数を指定する	4-15
ScGetDataReadyCount	データレディイベントを発生させる点数を取得する	4-15
ScAddEventListener	イベントリスナを登録する（C++ のみ）	4-16
ScRemoveEventListener	イベントリスナの登録を解除する（C++ のみ）	4-16
ScAddCallback	コールバックメソッドを登録する（C# のみ）	4-17
ScRemoveCallback	コールバックメソッドの登録を解除する（C# のみ）	4-17

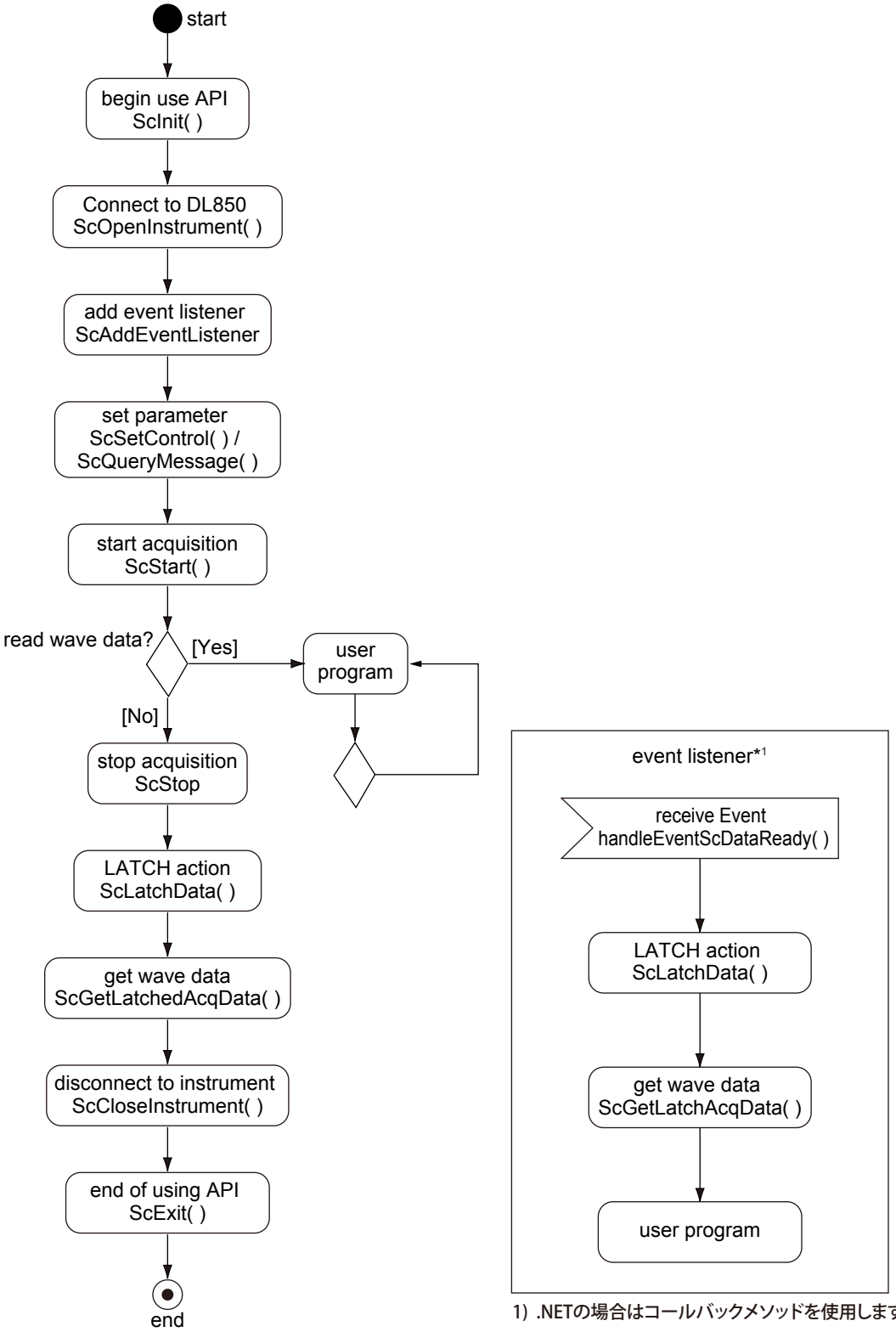
波形データの取得

フリーラン波形データの取得に関する API を以下に示します。

API Name	Function	Page
ScLatchData	測定位置をラッチする	4-8
ScGetLatchAcqData	ラッチ後に波形データを取得する	4-10

3.3 API 利用の流れ

各 API はハンドルベースで利用します。最初に機器に接続する時に接続ハンドルを生成し、その後そのハンドルを API の引数に渡すことで、指定機器へのアクセスを行います。



1) .NETの場合はコールバックメソッドを使用します。

アンマネージアプリケーション

以下に API 利用の流れと C++（アンマネージアプリケーション）を用いたコードの記述例を示します。なお、エラー処理は省略しています。

1. API の初期化（必ず必要です）

```
#include "ScAPI.h"
. . .
ScInit();
. . .
```

2. 測定器（DL850E/DL850EV）に接続して測定器ハンドルを生成（必ず必要です）
   接続後はこの測定器ハンドルを使って測定器にアクセスします。

```
ScHandle handle;
ScOpenInstrument(SC_WIRE_USB,"91K225903",&handle);
```

3. イベントリスナの登録
   データレディイベントを使う場合は、ScEventListener クラスを継承したクラス作成し、API に登録します。handleEventScDataReady() メソッドをオーバーライドすると、データレディイベント発生のタイミングで同メソッドが呼び出されます。イベントリスナの作成、登録は必須ではありません。（定周期で波形取得処理を呼び出しても波形取得は可能です）

```
class cYourClass : public ScEventListener {
public:
  virtual void handleEventScDataReady(ScHandl handle,
                                       __int64 dataCount);
};
. . .
cYourClass* yourClass = new cYourClass();
ScAddEventListener(handle, yourClass);
```

4. 測定開始

```
ScStart(handle);
```

5. ラッチ（波形を取得する場合は必ず必要です）
   波形データの取得位置をマークします。

```
ScLatchData(handle);
```

6. 波形データ取得

```
char buff[100000];
ScGetLatchAcqData(handle, 1, 0, buff, sizeof(buff), &count, &dataSize);
. . .
```

7. 測定器との通信を切断（必ず必要です）
   この API を呼び出した後、測定器ハンドルは無効になります。

```
ScCloseInstrument(handle);
```

8. API の使用終了（必ず必要です）

```
ScExit();
```

マネージアプリケーション

以下に API 利用の流れと C#（マネージアプリケーション）を用いたコードの記述例を示します。なお、エラー処理は省略しています。

1. API の初期化（必ず必要です）
   事前に Visual Studio のソリューションエクスプローラー「参照設定」で ScAPINet.dll を登録してください。名前空間は ScAPINet、API は ScAPI クラス内のメソッドとして定義されています。

```
using ScAPINet;
. . .
ScAPI api = new ScAPINet.ScAPI();
api.ScInit();
```

2. 測定器（DL850E/DL850EV）に接続して測定器ハンドルを生成（必ず必要です）
   接続後はこの測定器ハンドルを使って測定器にアクセスします。

```
int handle;
api.ScOpenInstrument(ScAPI.SC_WIRE_USB,"91K225903",out handle);
```

3. イベントコールバックの登録
   データレディイベントを使う場合は、コールバックメソッドを API に登録します。データレディイベント発生のタイミングで同メソッドが呼び出されます。コールバック作成、登録は必須ではありません。（定周期で波形取得処理を呼び出しても波形取得は可能です）

```
private void dataReadyCallback(int hndl, int type)
{
  . . .
}
api.ScAddCallback(hndl, dataReadyCallback);
```

4. 測定開始

```
api.ScStart(handle);
```

5. ラッチ（波形を取得する場合は必ず必要です）
   波形データの取得位置をマークします。

```
api.ScLatchData(handle);
```

6. 波形データ取得

```
byte[] buff = new byte[100000];
int count, dataSize;
api.ScGetLatchAcqData<byte>(handle, 1, 0, buff, buff.Length,
                            out count, out dataSize);
```

7. 測定器との通信を切断（必ず必要です）
   この API を呼び出した後、測定器ハンドルは無効になります。

```
api.ScCloseInstrument(handle);
```

8. API の使用終了（必ず必要です）

```
api.ScExit();
```

4.1　クラス定義

API のクラス定義について説明します。

Class ScEventListener

機能 :

イベントを受け取るためのイベントリスナークラス（C++ のみ）

書式 :

```
class ScEventListener {
    public:
        virtual void handleEventScDataReady(ScHandle handle,
        __int64 dataCount);
};
```

詳細 :

データレディイベントを受け取る場合は handleEventScDataReady() メソッドをオーバーライドします。ScAddEventListener() でインスタンスを登録します。

4.2 固定値定義

SC_SUCCESS

概要 :

正常

書式 :

[C++] #define SC_SUCCESS 0
[C#] ScAPI.SC_SUCCESS

詳細 :

API の各関数の返却値定義

SC_ERROR

概要 :

エラー

書式 :

[C++] #define SC_ERROR 1
[C#] ScAPI.SC_ERROR

詳細 :

API の各関数の返却値定義

SC_WIRE_USB

概要 :

USB 回線（USBTMC）

書式 :

[C++] #define SC_WIRE_USB 7
[C#] ScAPI.SC_WIRE_USB

詳細 :

DL850 シリーズに接続する時の回線種別定義

SC_WIRE_LAN

概要 :

LAN 回線（VXI-11）

書式 :

[C++] #define SC_WIRE_LAN 8
[C#] ScAPI.SC_WIRE_LAN

詳細 :

DL850 シリーズに接続する時の回線種別定義

4.3 API 詳細仕様

API の詳細仕様について説明します。

ScInit

概要 :

API を初期化する

書式 :

[C++] ScResult ScInit(void);
[C#] int ScInit();

引数 :

なし

戻り値 :

SC_SUCCESS 成功
SC_ERROR 初期化エラー (すでに初期化されている)

詳細 :

ライブラリの使用開始時に一度だけ呼び出します。

使用例 :[C++]

```
#include "ScAPI.h"
...
if (ScInit() == SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
using ScAPINet;
...
ScAPINet.ScAPI api = new ScAPINet.ScAPI();
if (api.ScInit() == ScAPI.SC_SUCCESS)
{
    ...
}
```

ScExit

概要 :

API の使用を終了する

書式 :

[C++] ScResult ScExit(void);
[C#] int ScExit();

引数 :

なし

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー (すでに終了しているか、初期化されていない)

詳細 :

API の使用終了時に一度だけ呼び出します。

ScOpenInstrument

概要 :

測定器に接続する

書式 :

[C++] ScResult ScOpenInstrument(int wire, char* address, ScHandle* rHndl);
[C#] int ScOpenInstrument(int wire, string address, out int rHndl);

引数 :

[IN] wire 回線種別
SC_WIRE_USB USBTMC 接続
SC_WIRE_LAN VXI-11
[IN] address 接続先アドレス（USB の場合は測定器シリアル番号）
[OUT] rHndl 測定器ハンドル

戻り値 :

SC_SUCCESS 接続成功
SC_ERROR 接続エラー

詳細 :

測定器に接続し、測定器ハンドルを返します。
各 API にはこのハンドルを渡して測定器と通信を行います。
接続時、測定器を自動的にフリーランモードに設定します。

注意 :

ひとつの測定器に多重に接続することはできません。

使用例 :[C++]

```
ScHandle hndl;
if (ScOpenInstrument(SC_WIRE_USB, "91K225895", &hndl)
    == SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
int hndl;
if (api.ScOpenInstrument(ScAPI.SC_WIRE_USB, "91K225895",
    out hndl) == ScAPI.SC_SUCCESS)
{
    ...
}
```

ScCloseInstrument

概要 :

測定器から切断する

書式 :

[C++] ScResult ScCloseInstrument(ScHandle hndl);
[C#] int ScCloseInstrument(int hndl);

引数 :

[IN] handle 測定器ハンドル

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー (接続していないか、切断済み)

詳細 :

ScOpenInsturument() で接続した測定器から切断します。
切断時、測定器を自動的にフリーランモードからトリガモードに戻します。

注意 :

この API を呼び出した後、ハンドルは無効になります。

ScSetControl

概要 :

通信コマンドを送信する

書式 :

[C++] ScResult ScSetControl(ScHandle hndl, char* command);
[C#] int ScSetControl(int hndl, string command);

引数 :

[IN] hndl 測定器ハンドル
[IN] command 通信コマンド文字列

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

通信コマンドを測定器に送信します。

注意 :

戻り値で通信コマンドのエラーは判定できません。正常に送信できたかどうかを示します。

ScGetControl

概要 :

通信コマンドの応答を受信する

書式 :

[C++] ScResult ScGetControl(ScHandle hndl, char* buff, int buffLen, int* receiveLen);
[C#] int ScGetControl<DT>(int hndl, ref DT[] buff, int buffLen, out int receiveLen);

引数 :

[IN] hndl 測定器ハンドル
[OUT] buff データ受信バッファ
[IN] buffLen バッファのサイズ
[OUT] receiveLen 受信した応答の長さ

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー (受信するデータがない)

詳細 :

測定器から事前に送信した通信コマンドの応答を受信します。

注意 :

事前に通信コマンドが送信されていない場合はエラーになります。

使用例 :[C++]

```
char buff[BUFSIZ];
int receiveLen;
if (ScGetControl(hndl, buff, sizeof(buff), &receiveLen)
    == SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
byte[] buff = new byte[256];
int receiveLen;
if (api.ScGetControl<byte>(hndl, ref buff, buff.Length,
    out receiveLen) == ScAPI.SC_SUCCESS)
{
    string msg = System.Text.Encoding.ASCII.GetString(buff);
    printMessage(msg);
}
```

ScGetBinaryData

概要 :

バイナリデータを受信する

書式 :

[C++] ScResult ScGetBinaryData(ScHandle hndl, char* command, char* buff, int buffLen, int* receiveLen);

[C#] int ScGetBinaryData<DT>(int hndl, string command, DT[] buff, int buffLen, out int receiveLen);

引数 :

[IN] hndl	測定器ハンドル
[IN] command	バイナリデータを要求する通信コマンド
[IN] buff	バイナリデータを受信するバッファ
[IN] buffLen	バイナリデータを受信するバッファのサイズ（バイト数）
[OUT] receiveLen	受信したバイナリデータのサイズ（バイト数）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

バイナリデータを返すコマンドを送信し、応答を受信します。

注意 :

バイナリデータを送信しないコマンドを指定した場合の動作は不定です。

使用例 :[C++]

```
char buff[1024];
int receiveLen;
if (ScGetBinaryData(hndl, ":MONitor:SEND:ALL?",
    buff, sizeof(buff), &receiveLen)== SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
byte[] buff = new byte[1024];
int receiveLen;
if (api.ScGetBinaryData<byte>(hndl, ":MONitor:SEND:ALL?",
    ref buff, buff.Length, out receiveLen) == ScAPI.SC_SUCCESS)
{
    ...
}
```

ScQueryMessage

概要 :

通信コマンドを送信しその応答を受信する

書式 :

[C++] ScResult ScQueryMessage(ScHandle hndl, char* command, char* buff, int buffLen, int* receiveLen);

[C#] int ScQueryMessage(int hndl, string command, out string buff, int getLen, out int receiveLen);

引数 :

[IN] hndl	測定器ハンドル
[IN] command	通信コマンド
[OUT] buff	受信バッファ
[IN] buffLen	受信バッファの長さ（バイト数）/.NET 版の場合は取得する長さ
[OUT] receiveLen	受信した応答の長さ

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

通信コマンドの送信と、応答の受信をひとつの API で行うことができます。

注意 :

応答を返さないコマンドにこの API を使うことはできません。
C#(.NET 版) の場合は 4 番目の引数に受信バッファの長さではなく、受信するバイト数を指定します。

使用例 :[C#]

```
char buff[256];
int receiveLen;
if (ScQueryMessage(hndl, "*idn?", buff, sizeof(buff), &receiveLen)
    == SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
string buff;
int receiveLen;
if (api.ScQueryMessage(hndl, "*idn?", out buff, 256,
    out receiveLen) == ScAPI.SC_SUCCESS)
{
    ...
}
```

ScStart

概要 :

測定を開始する

書式 :

[C++] ScResult ScStart(ScHandle hndl)
[C#] int ScStart(int hndl)

引数 :

[IN] hndl 測定器ハンドル

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

測定を開始します。(「Start」コマンドを送信します)

ScStop

概要 :

測定を停止する

書式 :

[C++] ScResult ScStop(ScHandle hndl)
[C#] int ScStop(int hndl)

引数 :

[IN] hndl 測定器ハンドル

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

測定を停止します。(「Stop」コマンドを送信します)

ScLatchData

概要 :

フリーランデータをラッチする

書式 :

[C++] ScResult ScLatchData(ScHandle hndl)
[C#] int ScLatchData(int hndl)

引数 :

[OUT] hndl 測定器ハンドル

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

測定器内のフリーラン測定データの現在の測定位置をマーキングします。
測定データを取得する際、このマーク位置を基準に取得することになります。

ScGetLatchCount

概要 :

ラッチ位置を取得する

書式 :

[C++] ScResult ScGetLatchCount(ScHandle hndl, __int64* count)
[C#] int ScGetLatchCount(int hndl, out long count)

引数 :

[IN] hndl 測定器ハンドル
[OUT] count ラッチ位置（点数）

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

ラッチ位置を取得します。
ラッチ位置は測定を開始してから ScLatchData() でラッチを実行した位置までのサンプル点数です。

注意 :

サンプル点数は 2ch モジュール使用の有無にかかわらず、2ch モジュール相当で取得したデータ点数を返します。

ScGetLatchIntervalCount

概要 :

ラッチ間の点数を取得する

書式 :

[C++] ScResult ScGetLatchIntervalCount(ScHandle hndl, __int64* count)
[C#] int ScGetLatchIntervalCount(int hndl, out long count)

引数 :

[IN] hndl 測定器ハンドル
[OUT] count ラッチ間の点数

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

前回ラッチした位置から今回ラッチした位置までのサンプル数を取得します。

注意 :

ラッチ間の点数は 2ch モジュール使用の有無にかかわらず、2ch モジュール相当で取得したデータ点数を返します。

ScGetLatchAcqData

概要 :

ラッチ済みの測定データを取得する

書式 :

[C++] ScResult ScGetLatchAcqData(ScHandle hndl, int chNo, int subChNo, char* buff,int buffLen, int* dataCount, int* dataSize);

[C#] int ScGetLatchAcqData<DT>(int hndl, int chNo, int subChNo, DT[] buff, int buffLen, out int dataCount, out int dataSize)

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号
[IN] subChNo	サブチャネル番号（無い場合は 0 を指定）
[OUT] buff	データを保存するバッファ
[IN] buffLen	データを保存するバッファの長さ
[OUT] dataCount	保存したデータの長さ（点数）
[OUT] dataSize	保存したデータ 1 点のサイズ（バイト数）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ラッチ済みの測定データを取得します。

注意 :

返却される測定データは AD 値です。
物理値に変換するには、ScGetChannelGain() で取得した Gain 値を掛け、ScGetChannelOffset() で取得した Offset 値を足す必要があります。

使用例 :[C++]

```
char buff[100000];
int count;
int size;
if (ScGetLatchAcqData(hndl, 1, 0, buff, sizeof(buff),
    &count, &size) == SC_SUCCESS) {
    ...
}
```

使用例 :[C#]

```
byte[] buff = new byte[100000];
int count;
int size;
if (api.ScGetLatchAcqData<byte>(hndl, 1, 0, buff, buff.Length,
    out count, out size)== ScAPI.SC_SUCCESS)
{
    ...
}
```

ScGetChannelDelay

概要 :

チャネルの位相差を取得する

書式 :

[C++] ScResult ScGetChannelDelay(ScHandle hndl, int chNo, int* delay)
[C#] int ScGetChannelDelay(int hndl, int chNo, out int delay)

引数 :

[IN] hndl 測定器ハンドル
[IN] chNo チャネル番号
[OUT] delay 位相差

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

チャネルの位相差を取得します。
対象チャネルにサブチャネルがある場合、サンプルレート比に応じて位相差が発生する場合があります。
この API はその位相差点数を返します。

注意 :

多チャンネルモジュールのサブチャネルごとの位相差は同じです。

ScGetStartTime

概要 :

測定開始時刻を取得する

書式 :

[C++] ScResult ScGetStartTime(ScHandle hndl, char* buff);
[C#] int ScGetStartTime(int hndl, out string buff)

引数 :

[IN] hndl 測定器ハンドル
[OUT] buff 測定開始時刻文字列

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

測定開始時刻を文字列で取得します。
時刻はカンマで区切られた文字列として返します。
年 (2007 ～), 月 (1 ～ 12), 日 (1 ～ 32), 時 (0 ～ 23), 分 (0 ～ 59), 秒 (0 ～ 59), マイクロ秒 (0 ～ 999999), ナノ秒 (10 ～ 990)

注意 :

停止中に呼び出した場合は、前回測定を開始した時刻を返します。

ScSetSamplingRate

概要 :

サンプリング周波数を設定する

書式 :

[C++] ScResult ScSetSamplingRate(ScHandle hndl, double srate);
[C#] int ScSetSamplingRate(int hndl, double srate)

引数 :

[IN] hndl 測定器ハンドル
[IN] srate サンプリング周波数 (Hz)

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

サンプリング周波数を設定します。

注意 :

測定中は設定できません。

ScGetSamplingRate

概要 :

サンプリング周波数を取得する

書式 :

[C++] ScResult ScGetSamplingRate(ScHandle hndl, double* srate)
[C#] int ScGetSamplingRate(int hndl, out double srate)

引数 :

[IN] hndl 測定器ハンドル
[OUT] srate サンプリング周波数

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

サンプリング周波数を取得します。

ScGetBaseSamplingRate

概要 :

基本サンプル周波数を取得する

書式 :

[C++] ScResult ScGetBaseSamplingRate(ScHandle hndl, double* srate)
[C#] int ScGetBaseSamplingRate(int hndl, out double srate)

引数 :

[IN] hndl 測定器ハンドル
[OUT] srate サンプル周波数

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

基本サンプル周波数（2ch モジュールのサンプル周波数）を取得します。

ScGetChannelSamplingRatio

概要 :

チャネルのサンプル周波数と基本サンプル周波数の比率を取得する

書式 :

[C++] ScResult ScGetChannelSamplingRatio(ScHandle hndl, int chNo, int* ratio)
[C#] int ScGetChannelSamplingRatio(int hndl, int chNo, out int ratio)

引数 :

[IN] hndl 測定器ハンドル
[IN] chNo チャネル番号（1 ～ 16）
[OUT] ratio サンプル周波数比率（1 ～ 1000）

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

チャネルのサンプル周波数と、基本サンプル周波数の比率を取得します。
基準のサンプル周波数と同じであれば「1」となり、半分であれば「2」となります。
サブチャネルのあるチャネルの場合、サンプル周波数が基本サンプル周波数（2CH モジュールのサンプル周波数）よりも低い場合があり、またサンプル点数も同様に比率に応じて少なくなります。

ScGetChannelBits

概要 :

チャネルのデータビット長を取得する

書式 :

[C++] ScResult ScGetChannelBits(ScHandle hndl, int chNo, int subChNo, int* bits);
[C#] int ScGetChannelBits(int hndl, int chNo, int subChNo, out int bits)

引数 :

[IN] hndl 測定器ハンドル
[IN] chNo チャネル番号（1 ～ 16）
[IN] subChNo サブチャネル番号（1 ～ 64）
[OUT] bits データビット長（1 ～ 32）

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

取得するチャネルデータのビット長を取得します。

注意 :

CAN モジュールなどの場合、返却される値は必ずしも「Bit Cnt」で指定したビット数と同じにはなりません。

ScGetChannelGain

概要 :

チャネルのゲイン値を取得する

書式 :

[C++] ScResult ScGetChannelGain(ScHandle hndl, int chNo, int subChNo, double* gain);
[C#] int ScGetChannelGain(int hndl, int chNo, int subChNo, out double gain)

引数 :

[IN] hndl 測定器ハンドル
[IN] chNo チャネル番号（1 ～ 16）
[IN] subChNo サブチャネル番号（1 ～ 64/ 無い場合は 0 を指定する）
[OUT] gain ゲイン値

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

取得する測定データを物理値に変換する場合に利用するゲイン値を取得します。

ScGetChannelOffset

概要 :

チャネルのデータオフセットを取得する

書式 :

[C++] ScResult ScGetChannelOffset(ScHandle hndl, int chNo, int subChNo, double* offset);
[C#] int ScGetChannelOffset(int hndl, int chNo, int subChNo, out double offset)

引数 :

[IN] hndl 測定器ハンドル
[IN] chNo チャネル番号（1 ～ 16）
[IN] subChNo サブチャネル番号（1 ～ 64/ 無い場合は 0 を指定する）
[OUT] offset オフセット値

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

取得する測定データを物理値に変換する場合に利用するオフセット値を取得します。

ScSetDataReadyCount

概要 :

DataReady イベントを発生させる測定点数を設定する

書式 :

[C++] ScResult ScSetDataReadyCount(ScHandle hndl, int sampleCount)
[C#] int ScSetDataReadyCount(int hndl, int sampleCount)

引数 :

[IN] hndl 測定器ハンドル
[IN] sampleCount サンプル点数

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

フリーラン測定中に、一定の点数を測定するごとにデータレディイベントを発生させることができます。
データレディイベントを発生させる測定点数を指定します。
サンプリング周波数と同じ値（サンプリング周波数が 100kHz の場合、100,000）を指定すると、1 秒ごとにイベントが発生します。

ScGetDataReadyCount

概要 :

DataReady イベントを発生させる測定点数を取得する

書式 :

[C++] ScResult ScGetDataReadyCount(ScHandle hndl, int* sampleCount)
[C#] int ScGetDataReadyCount(int hndl, out int sampleCount)

引数 :

[IN] hndl 測定器ハンドル
[OUT] sampleCount サンプル点数

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

データレディイベントを発生させる測定点数を取得します。

ScAddEventListener

概要 :

イベントリスナーを登録する

書式 :

[C++] ScResult ScAddEventListener(ScHandle hndl, ScEventListener* listener)

引数 :

[IN] hndl 測定器ハンドル
[IN] listener イベントリスナークラスへのポインタ

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

ScEventListener クラスを継承したクラスをイベントリスナークラスとして登録することができます。
handleEventScDataReady() をオーバーライドすることによって、データレディイベント発生時に同メソッドが自動的に呼び出されます。

注意 :

現在、捕捉できるイベントは「データレディイベント」のみです。
handleEventScDataReady() 呼び出し時に引数に渡される dataCount は前回値です。
.NET 版（C#）では使えません。

使用例 :

```
class cMyEvent : public ScEventListener {
public:
    virtual void handleEventScDataReady(ScHandle hndl,
    __int64 dataCount);
};

cMyEvent* ep = new cMyEvent();
ScAddEventListener(hndl, ep);
```

ScRemoveEventListener

概要 :

イベントリスナーの登録を解除する

書式 :

[C++] ScResult ScRemoveEventListener(ScHandle hndl, ScEventListener* listener);

引数 :

[IN] hndl 測定器ハンドル
[IN] listener イベントリスナークラスへのポインタ

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

登録したイベントリスナーの登録を解除します。

注意 :

登録していないイベントリスナーを指定するとエラーになります。
.NET 版（C#）では使えません。

ScAddCallback

概要 :

コールバックメソッドを登録する（C# のみ）

書式 :

[C#] public delegate void ScCallback(int hndl, int type)
int ScAddCallback(int hndl, ScCallback func)

引数 :

[IN] hndl 測定器ハンドル
[IN] func コールバックメソッド

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

データレディイベント発生時に呼び出されるコールバックメソッドを登録します。

注意 :

現在、捕捉できるイベントは「データレディイベント」のみです。
C++ では使えません。
コールバックメソッドの type にはイベントの種別が渡されますが、現在未使用です。
使用例 :

```
private void dataReadyCallback(int hndl, int type)
{
    ....
}
if (api.ScAddCallback(hndl, dataReadyCallback) != ScAPI.SC_SUCCESS)
{
    // error
}
```

ScRemoveCallback

概要 :

コールバックメソッドの登録を解除する（C# のみ）

書式 :

[C#] int ScRemoveCallback(int hndl, ScCallback func)

引数 :

[IN] hndl 測定器ハンドル
[IN] func コールバックメソッド

戻り値 :

SC_SUCCESS 成功
SC_ERROR エラー

詳細 :

データレディイベント発生時に呼び出されるコールバックメソッドの登録を解除します。

注意 :

C++ では使えません。

4.4 DLL リンク方式と配置

現在、C++ で利用の場合、DLL のリンク方法としては暗黙的なリンク (Implicit linking) のみを想定しています。
暗黙的リンクで API を利用するには、インポートライブラリ (.lib ファイル) を指定しリンクし、API は通常の関数と同様に呼び出してください。

また、作成したアプリケーション（exe）と同じフォルダに以下の DLL を配置してください。

プロジェクト	C++（アンマネージアプリ）		C#（マネージアプリ）		
アーキテクチャ	32bit	64bit	32bit	64bit	Any CPU
ScAPI.dll	○		○		○
ScAPI64.dll		○		○	○
ScAPINet.dll			○	○	○
tmctl.dll	○		○		○
tmctl64.dll		○		○	○
YKMUSB.dll	○		○		○
YKMUSB64.dll		○		○	○