User's
Manual

スコープコーダ SDK
ユーザーズマニュアル

YOKOGAWA
横河計測株式会社

IM D165-01JA
初版

はじめに

このユーザーズマニュアルは、スコープコーダ SDK 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。
ご使用前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、DL950/SL2000 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

- 性能・機能の向上などにより、本書の内容を予告なしに変更することがあります。最新のマニュアルは、当社 Web サイトでご確認ください。
- 本書に記載の画面表示内容は実際のもの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、お買い求め先か、当社支社・支店・営業所までご連絡ください。
- 本書の内容の全部または一部を無断で転載、複製することは禁止されています。

商標

- Microsoft、Windows、Windows 10、Windows 11、および Visual Studio 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。
- Adobe、Acrobat は、Adobe Inc.(アドビ社) の登録商標または商標です。
- 本文中の各社の登録商標または商標には、®、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

2025 年 6 月　初版発行

1st Edition: June 2025 (YMI)

ご使用にあたっての注意

使用上の注意

- 本ソフトウェアは、DL950/SL2000 シリーズの API ライブラリです。他の製品には使用できません。
- 本ソフトウェアのバージョン、および使用する DL950/SL2000 のファームウェアバージョンをご確認の上、ご使用ください。対応する DL950/SL2000 のファームウェアバージョンは Ver2.01 以降となります。

免責事項

当社は、お客様が本ソフトウエアをダウンロードしインストールされた時点で、下記の免責事項を許諾いただいたものとみなします。

- 本ソフトウェアをダウンロードしインストールすることによって生じるいかなる問題についても、当社はその責務を負いません。
- 本ソフトウェアの使用に関して、直接または間接に生じるいっさいの損害について、当社はその責務を負いません。
- 本ソフトウェアは無償で提供されますが、本製品になんの欠陥もないという無制限の保証をするものではありません。また、本ソフトウェアに関する不具合修正や質問についてのお問い合わせをお受けできない場合があります。
- 本ソフトウェアに関する財産権、所有権、知的財産権、その他一切の権限は、当社に帰属します。

目次

第 5 章　付録

1.1　ソフトウェア概要

概要

本ソフトウエア（スコープコーダ SDK）は、DL950/SL2000 シリーズのアクイジションにおける波形データ取得、フラッシュアクイジションに保存されているデータを PC に転送、およびファイル操作・ファイル転送の機能に関する API（Application Programming Interface）を提供するものです。

機能

本ソフトウェアを利用して、以下の機能を実現できます。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
- フラッシュアクイジションに保存されている波形データの測定条件の取得
- フラッシュアクイジションに保存されている波形データの取得
- 本体に保存されているファイルのリスト取得
- ファイルの操作及び転送（本体→ PC、PC →本体）

Note

本 API に記載されていない機能（主にチャネル（垂直軸）に関する設定）については、DL950 スコープコーダ /SL2000 高速データアクイジションユニット 通信インタフェース ユーザーズマニュアル（IM DL950-17JA）を参考にして、通信コマンドで実装してください。

ソフトウェア構成

本ソフトウェアは以下で構成されます。

フォルダ名	ファイル名	内容
	readme.txt	スコープコーダ SDK のバージョン情報
dll	ScSDK.dll	API 本体
	ScSDK64.dll	API 本体 64bit 版
	ScSDKNet.dll	.NET 版フリーラン API ライブラリ
	tmctl.dll	通信用ライブラリ（7.0.0.0）
	tmctl64.dll	通信用ライブラリ 64bit 版（7.0.0.0）
doc	IMD165-01JA_010.pdf	スコープコーダ SDK ユーザーズマニュアル（本書）
sample		サンプルプログラム ▶内容は、5.6 節「サンプルプログラム」を参照
vc	ScSDK.lib	API インポートライブラリ（C++ のみ）
	ScSDK64.lib	API インポートライブラリ 64bit 版（C++ のみ）
	ScSDK.h	関数プロトタイプ宣言のヘッダファイル（C++ のみ）

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。

Windows10（32bit、64bit）の日本語版または英語版が動作する PC

Windows11 の日本語版または英語版が動作する PC

なお、本ソフトウェアでフリーランによる波形の取り込みを行う場合、指定されたバッファーにデータを保存します。API で必要なメモリーサイズについては、5.1 節の「フリーランモードについて」の「必要なメモリーサイズについて」を参照してください。

開発環境

Visual Studio 2017 以降 .NET Framework 4.7 以降

ユーザプログラムを実行する場合の推奨環境について

本ソフトウェアを使用してお客様が作成したプログラムにてフリーランによる波形の取り込みを行う場合、波形の取り込み条件や通信環境に応じて次に記述するような環境が必要になる場合があります。

10Gbit Ethernet 通信利用時

- CPU
  デスクトップ PC
  Intel Core i7-1165G7 以上 / 4 コア（8 スレッド）以上 / 4.7GHz 以上
- メモリー
  16GByte 以上
- SSD
  512GB 以上（M.2 スロット SSD 推奨、読み書き性能 3GB/s 以上）

1Gbit Ethernet/USB 通信利用時

- CPU
  Intel Core i5-10210U 以上 / 4 コア（8 スレッド）以上 / 4.2GHz 以上
- メモリー
  8GByte 以上
- SSD
  256GB 以上（読み書き性能 400MB/s 以上）

USB ドライバについて

本ソフトウェアを USB 接続にて使用する場合、専用の USB ドライバ（YTUSB）もしくは IVI ドライバ（VISA）が必要です。最新の USB ドライバは、次の Web ページからダウンロードできます。

https://tmi.yokogawa.com/jp/library/

YTUSB フォルダ内の Setup.exe を実行してください。インストールウィザードが起動します。インストール方法の詳細は、YTUSB フォルダー内のマニュアル（ReadMe_ja.pdf）をご覧ください。

DL950/SL2000 ファームウェアのバージョンについて

本ソフトウェアは、ファームウェアのバージョンが Ver2.01 以降の DL950/SL2000 で使用できます。最新のファームウェアは弊社の Web ページからダウンロードをお願いします。

https://tmi.yokogawa.com/jp/library/

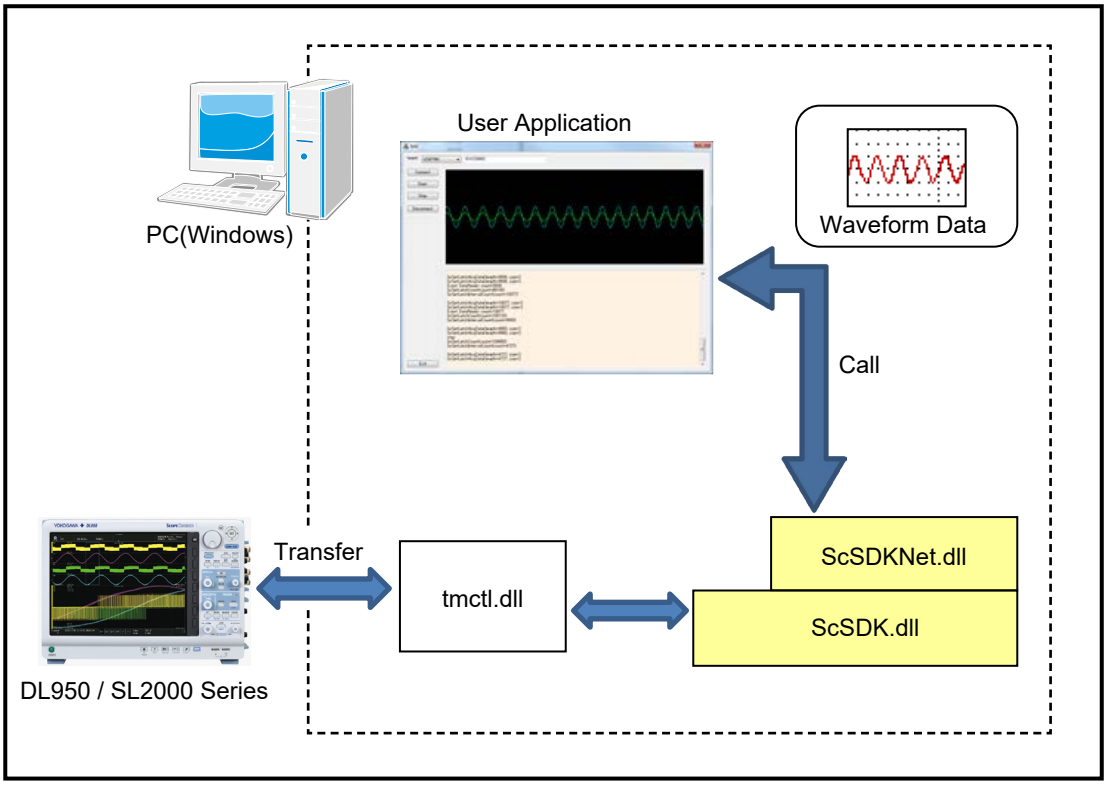
2.1　API 概要

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。
API は以下の 3 つの機能を提供します。

- データアクイジション機能
- フラッシュアクイジションデータアクセスライブラリ機能
- ファイル操作・転送機能

データアクイジション機能

データアクイジション機能は下図のように、アクイジショ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。（フリーラン測定およびトリガ測定）



本 API で提供されるデータアクイジション機能は、フリーランモードおよびトリガモードの 2 種類に対応しています。

（1）フリーランモード
波形の取り込みを開始してから停止するまでデータ取得を続けたい場合に使用します。

フリーランによる波形の取り込みの仕様

最大転送レート	320MB/S（10MS/s × 16ch）：10Gbit Ethernet 接続時
最大転送レート	6.4MB/S（200kS/s × 16ch）：1Gbit Ethernet/USB 接続時
最大波形取り込み時間	10 日間（※本 API で動作保証する最大時間になります）

※複数台同期接続による測定では、DL950/SL2000 からは上記転送レートでデータを送ると接続環境や PC 側のパフォーマンスなどの影響によりデータ送信用のバッファーオーバーランが発生する可能性が高くなります。複数台の転送レートの合計が上記範囲内で測定を行うことを強く推奨します。

（2）トリガモード
トリガによる波形取り込みを実施する場合に使用します。本 API を使用したトリガモードには、PC と DL950/SL2000 が同期をとりながら波形取り込みを行う同期モードと、DL950/SL2000 が PC とは非同期に取り込みを行う非同期モードの 2 種類のモードがあります。
なお、本 API では以下の機能には対応していません。

- ロールによる波形取り込み（※ DL950/SL2000 本体はロールによる波形取り込みは可能ですが、ロールによる波形取り込み中に本 API での波形取得には対応していません）
- DL950/SL2000 のトリガモードが N シングル
- DualCapture による波形取り込み
- リアルタイム記録（SSD 記録、フラッシュアクイジション）
- メモリーレコーダモード

トリガによる波形の取り込みの仕様
最大波形取り込み時間　10 日間（※本 API で動作保証する最大時間になります）
10GbpsEthernet を用いた高速転送モードが有効な場合、メモリー結合の制限により設定可能な最大レコード長は以下のようになります。メモリー結合の詳細については、「DL950 スコープコーダ /SL2000 高速データアクイジションユニット ユーザーズマニュアル [操作編]（IM DL950-03JA）」の付録をご覧ください。
標準モデル：250M
/M1 モデル：1G
/M2 モデル：2G

フラッシュアクイジションデータアクセスライブラリ機能

フラッシュアクイジションデータアクセスライブラリは、DL950/SL2000 に保存されているフラッシュアクイジションの波形データを本体に読み込みをすることなく直接 PC に取り出す機能をアプリケーションに提供します。

※ ただし、本 API を使用してフラッシュアクイジションの波形データの取り出しを行う際には、アクイジションメモリーを一時バッファーとして使用するため、メディアに保存されていないアクイジションメモリーにあるデータおよびヒストリ情報は消去されます。フラッシュアクイジションに保存されている波形データは影響しません。
※ 本機能は、/ST2 オプションが実装されている場合のみ有効です。

ファイル操作・転送機能

ファイル操作・転送機能は、DL950/SL2000 の測定データや設定の取得、送信、削除に関連する機能をアプリケーションに提供します。

2.2 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）	4-1
ScExit	API の使用終了	4-1

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	4-2
ScReopenInstrument	測定器に再接続する	4-3
ScCloseInstrument	測定器から切断する	4-4
ScOpenInstrumentEx	複数台同期の測定器に接続し、接続ハンドルのリストを取得する	4-4
ScCloseInstrumentEx	接続ハンドルリストを使用し測定器から切断する	4-7
ScSearchDevices	接続可能な測定器のリストを取得する	4-12

波形の取り込み条件の設定と取得

波形の取り込み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	4-7
ScGetControl	コマンド応答を測定器から受信する	4-8
ScQueryMessage	通信コマンドを送信し応答を受信する	4-10
ScGetBinaryData	バイナリデータを受信する	4-9
ScSetSamplingRate	サンプリングレートを設定する	4-27
ScGetSamplingRate	サンプリングレートを取得する	4-28
ScGetChannelSamplingRate	チャネルのサンプリングレートを取得する	4-28
ScSet10GMode	10G 高速転送モードを設定する	4-11
ScGet10GMode	10G 高速転送モードを取得する	4-12
ScStart	波形の取り込みを開始する	4-15
ScStop	波形の取り込みを停止する	4-15
ScGetStatusInfo	測定器のステータス情報を取得する	4-13
ScTmcSetTimeout	TMCTL のタイムアウト時間を設定する	4-13

トリガによる波形取り込み情報の取得

トリガによる波形取り込み情報の取得に関する API を以下に示します。

API Name	Function	Page
ScSetMeasuringMode	データアクイジションモードを設定する	4-14
ScGetLatchAcqCount	ラッチした時点の最新アクイジションカウントを取得する	4-23
ScGetAcqCount	データを取得するアクイジションカウントを取得する	4-24
ScSetAcqCount	データを取得するアクイジションカウントを設定する	4-24
ScGetTriggerTime	指定のアクイジションカウントのトリガ時刻を取得する	4-25
ScResumeAcquisition	同期モードによる波形取り込みを再開する	4-25
ScSetTriggerTimeout	同期モードの DL950/SL2000 側タイムアウト値を設定する	4-26
ScGetTriggerTimeout	同期モードの DL950/SL2000 側タイムアウト値を取得する	4-26

波形データの取得

フリーランでの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	波形の取り込み情報をラッチする	4-16
ScGetLatchRawData	ラッチ後に波形データを取得する	4-18
ScGetChAcqData	ScGetLatchRawData で取得したブロック形式のデータから指定チャネルの波形データの情報を取得する	4-19
ScGetFreeRunDataLength	フリーランモードでタイムベース基準の波形データ点数を取得する	4-23

トリガでの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	波形の取り込み情報をラッチする	4-16
ScGetAcqData	指定のアクイジションカウントの測定データを取得する	4-21
ScGetAcqDataLength	指定のアクイジションカウントのデータ長を取得する	4-22

波形データの変換

波形データの物理値への変換に関する API を以下に示します。

API Name	Function	Page
ScGetChannelBits	該当チャネル 1 データあたりのビット数を取得する	4-29
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）	4-29
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）	4-30
ScGetChannelScale	チャネルの表示スケールの上限値／下限値を取得する	4-30
ScGetChannelType	チャネルの波形データのデータ種別を取得する	4-31

イベントリスナー / コールバック関数

イベントリスナーおよびコールバックに関する API を以下に示します。

API Name	Function	Page
ScAddEventListener	イベントリスナーを登録する（C++ のみ）	4-31
ScRemoveEventListener	イベントリスナーの登録を解除する（C++ のみ）	4-32
ScAddCallback	コールバックメソッドを登録する（C# のみ）	4-33
ScRemoveCallback	コールバックメソッドの登録を解除する（C# のみ）	4-33

フラッシュアクイジションの波形データ情報の取得

フラッシュアクイジションの波形データ情報の取得に関する API を以下に示します。

API Name	Function	Page
ScGetFAcqCount	測定器に保存されているフラッシュアクイジションの波形データ数を取得する	4-34
ScGetFAcqFileName	測定器に保存されているフラッシュアクイジションの波形データのファイル名を取得する	4-34
ScOpenFAcqData	データ転送を行う波形データをオープンする	4-35
ScCloseFAcqData	オープンした波形データをクローズする	4-35
ScClearAcqMemory	アクイジションメモリーにある波形データを消去する	4-36
ScGetFAcqStartTime	オープンした波形データの測定開始時刻を取得する	4-36
ScGetFAcqTimeBase	オープンした波形データのタイムベース設定を取得する	4-37
ScGetFAcqComment	オープンした波形データのコメントを取得する	4-37
ScGetFAcqChannelCount	オープンした波形データに保存されているチャネル総数を取得する	4-38
ScGetFAcqChannelNumber	オープンした波形データに保存されているチャネル番号を取得する	4-38

波形データに保存されているチャネル情報の取得

オープンした波形データに保存されているチャネル情報の取得に関する API を以下に示します。

API Name	Function	Page
ScGetFAcqChannelBits	指定チャネルの 1 データあたりのビット数を取得する	4-39
ScGetFAcqChannelGain	指定チャネルのゲイン値を取得する	4-40
ScGetFAcqChannelHOffset	指定チャネルの測定開始時刻からオフセット時間を取得する	4-40
ScGetFAcqChannelHResolution	指定チャネルのサンプルインターバル値を取得する	4-41
ScGetFAcqChannelLabel	指定チャネルのラベル名を取得する	4-42
ScGetFAcqChannelLogicBits	指定チャネルが Logic の場合に有効なビット数を取得する	4-42
ScGetFAcqChannelLogicLabel	指定チャネルが Logic の場合に、各ビットごとのラベル名を取得する	4-43
ScGetFAcqChannelOffset	指定チャネルのオフセット値を取得する	4-43
ScGetFAcqChannelSign	指定チャネルの符号情報を取得する	4-44
ScGetFAcqChannelType	指定チャネルのデータ種別を取得する	4-45
ScGetFAcqChannelUnit	指定チャネルの単位文字列を取得する	4-45

波形データの取得

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
ScSetFAcqChannelNumber	波形取得するチャネル番号を設定する	4-46
ScGetFAcqDataLength	波形取得するデータ点数の取得する	4-46
ScGetFAcqData	指定したチャネルのデータを取得する	4-47
ScSetFAcqDataSize	一回で送信されるデータの最大サイズを設定する	4-48
ScSet10GMode	10Gbps 高速転送モードを設定する	4-11
ScGet10GMode	10Gbps 高速転送モードを取得する	4-12

ファイル操作・転送

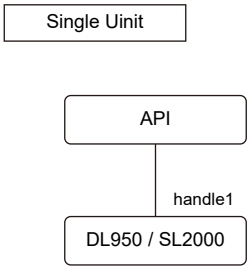
ファイル操作・転送に関連する API を以下に示します。

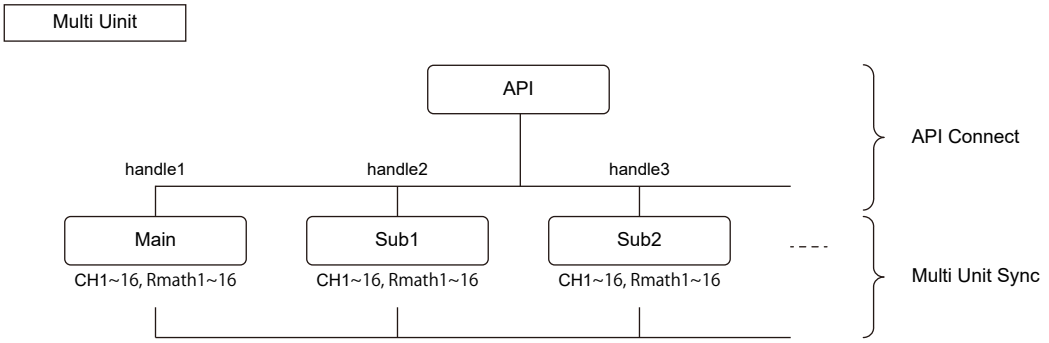
API Name	Function	Page
ScSetCurrentDrive	カレントドライブを設定する	4-49
ScGetCurrentDrive	カレントドライブを取得する	4-49
ScSetCurrentDirectory	カレントディレクトリを設定する	4-50
ScGetCurrentDirectory	カレントディレクトリを取得する	4-50
ScGetFileNum	ファイル数を取得する	4-51
ScGetFileInfo	ファイル情報を取得する	4-51
ScDeleteFile	ファイルを削除する	4-52
ScDownloadFile	測定器のファイルを取得する	4-52
ScUploadFile	ファイルを測定器に保存する	4-53
ScSaveTriggerWDF	トリガ波形を WDF ファイルとして取得する	4-53
ScSaveFreeRunWDF	フリーラン波形を WDF ファイルとして取得する	4-54
ScSaveSetup	測定器の設定を取得する	4-54
ScLoadSetup	設定ファイルの設定を測定器に反映する	4-55
ScGetFileList	ファイルリストを取得する	4-55
ScGetFileInfoList	ファイル情報リストを取得する	4-57

2.3 API 利用の流れ

ハンドル

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ときに接続ハンドルを生成し、そのハンドルを API の引数に渡すことで、指定の機器へアクセスを行います。
複数台同期接続した機器に接続する場合は、1 つの機器に対し 1 つのハンドルを生成することで、複数台同期の機器へアクセスを行います。





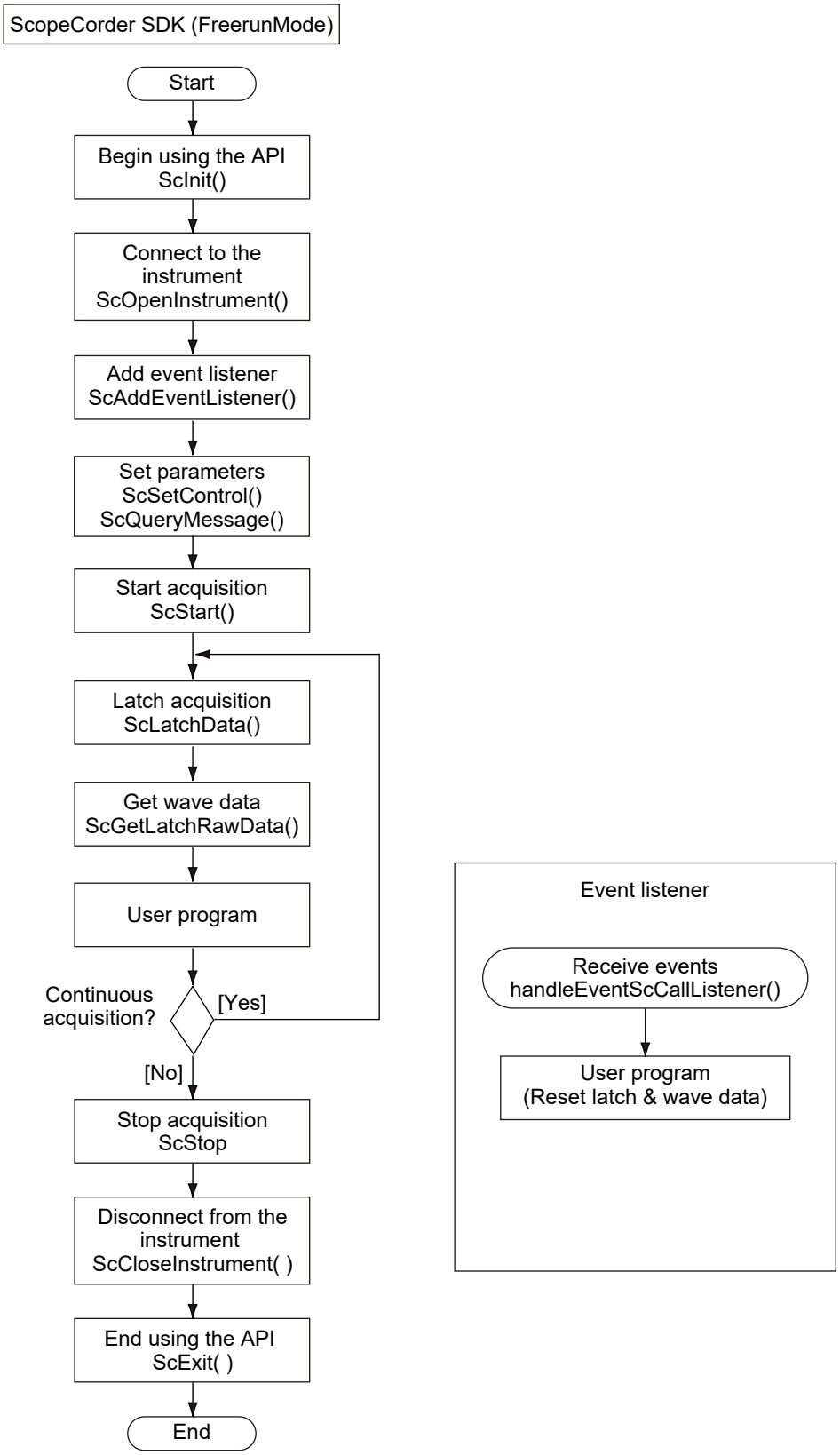
設定、クエリ

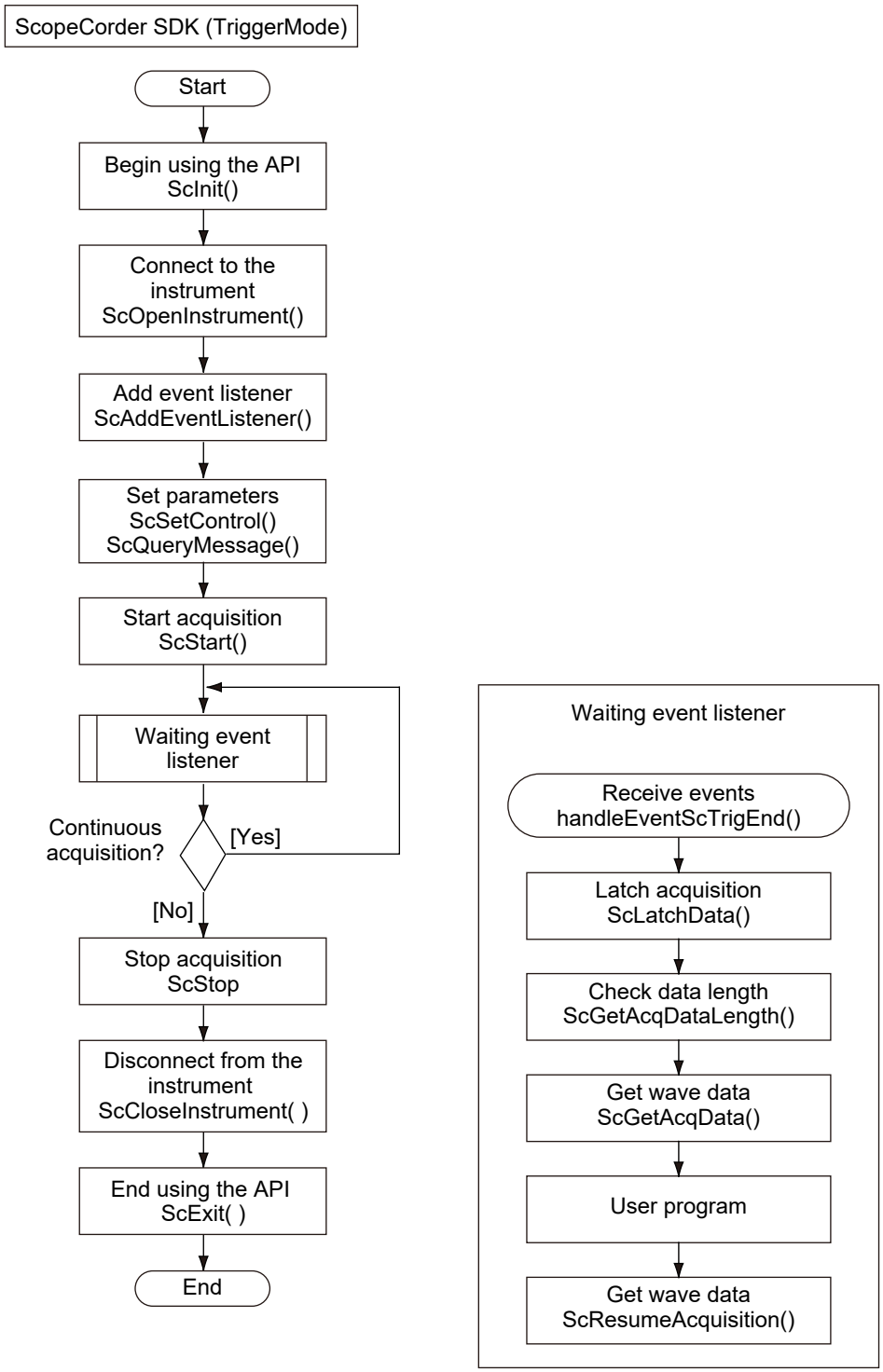
API はハンドルによってデータアクイジション、フラッシュアクイジションのデータアクセスおよびファイル操作・転送を行います。それ以外の機能を使用する場合は DL950/SL2000 の通信コマンドを ScSetControl や ScGetControl 等の API を用いて送受信します。詳細は 5.3 節をご覧ください。また、本 API はコマンドに対する応答がすべてヘッダーなしのデータのみとなるよう、接続開始時に設定しています。

イベントメッセージ

API には、アプリケーションプログラムの実行効率を高めるために、機器でのトリガ検出や測定終了したことをイベントメッセージで通知する機能があります。ここではこれをイベントと呼びます。イベントは機器からの SRQ（サービスリクエスト）割り込みを利用して発生させています。これにより、通知されるイベントに応じて処理を行なうプログラムを記述することができます。

データアクイジション機能




ScopeCorder SDK (TriggerMode)
Start
Begin using the API
ScInit()
Connect to the instrument
ScOpenInstrument()
Add event listener
ScAddEventListener()
Set parameters
ScSetControl()
ScQueryMessage()
Start acquisition
ScStart()
Waiting event listener
Continuous acquisition?
[Yes]
[No]
Stop acquisition
ScStop
Disconnect from the instrument
ScCloseInstrument( )
End using the API
ScExit( )
End
Waiting event listener
Receive events
handleEventScTrigEnd()
Latch acquisition
ScLatchData()
Check data length
ScGetAcqDataLength()
Get wave data
ScGetAcqData()
User program
Get wave data
ScResumeAcquisition()

アンマネージアプリケーション（フリーランモード）

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 "ScSDK.h"
. . .
ScInit();
. . .
```

2. 測定器（DL950/SL2000）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", SC_FREERUN,
 &handle);
```

3. イベントリスナーの登録
   フリーランモードでは 10GEther 接続以外の I/F を使用している場合に、データのオーバーランを検出することができます。オーバーランを検出する場合は、オーバーランイベントを使用します。オーバーランイベントを使用するには、ScEventListener クラスを継承したクラスを作成し、API に登録します。handleEventScCallListener メソッドをオーバーライドすると、オーバーランの発生のタイミングで同メソッドが呼び出されます。フリーランでオーバーランを検出した場合、波形データ取得で取得されるデータが無効（送られてくるデータは保証されません）となります。この場合、ラッチコマンドを連続して送ることで状態を解除することが可能です。
   なお、波形の取り込みのサンプリングが遅い場合でかつ通信環境としてデータを連続して取得できるような場合には、オーバーラン検出を登録しなくても波形取り込みは可能です。

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

4. 波形の取り込み開始

```
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];
ScGetLatchRawData(handle, buff, sizeof(buff), &recieveLen);
. . .
```

   波形取り込み中は、5（ラッチ）、6（波形データ取得）を繰り返します。

7. 波形の取り込み停止

```
ScStop(handle);
```

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

```
ScCloseInstrument(handle);
```

9. 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（フリーランモード）

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 ScSDKNet.dll を登録してください。名前空間は ScSDKNet、API は ScSDK クラス内のメソッドとして定義されています。

```
using ScSDKNet;
. . .
ScSDK api = new ScSDKNet.ScSDK();
api.ScInit();
```

2. 測定器 DL950/SL2000）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(ScSDK.SC_WIRE_USB, "91K225903",
 ScSDK.SC_FREERUN, out handle);
```

3. イベントコールバックの登録
   フリーランモードでは 10GEther 接続以外の I/F を使用している場合に、データのオーバーランを検出することができます。オーバーランの検出を行う場合は、オーバーランイベントを使用します。オーバーランイベントを使用するには、コールバックメソッドを API に登録します。オーバーランの発生のタイミングで同メソッドが呼び出されます。フリーランでオーバーランを検出した場合、波形データ取得で取得されるデータが無効（送られてくるデータは保証されません）となります。この場合、ラッチコマンドを連続して送ることで状態を解除することが可能です。
   なお、波形の取り込みのサンプリングが遅い場合でかつ通信環境としてデータを連続して取得できるような場合には、オーバーラン検出を登録しなくても測定は可能です。

```
private void overrunCallback(int hndl, int type)
{
  . . .
}
api.ScAddCallback(hndl, overrunCallback,
 ScSDK.SC_EVENTTYPE_OVERRUN);
```

4. 波形の取り込み開始

```
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 recieveLen;
api.ScGetLatchRawData<byte>(handle, buff, buff.Length,
 out receiveLen);
```

   波形取り込み中は、5（ラッチ）、6（波形データ取得）を繰り返します。

7. 波形の取り込み停止

```
api.ScStop(handle);
```

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

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

9. API の使用終了（必ず必要です）

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

アンマネージアプリケーション（トリガモード）

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 "ScSDK.h"
. . .
ScInit();
. . .
```

2. 測定器（DL950/SL2000）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", SC_TRIGGER,
 &handle);
```

3. イベントリスナーの登録
   トリガの同期モードの場合は、トリガエンドイベントを使用します。トリガエンドイベントを使用するには、ScEventListener クラスを継承したクラスを作成し、API に登録します。handleEventScTrigEnd メソッドをオーバーライドすると、トリガエンド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。
   トリガ非同期モードの場合、イベントが発生しないためイベントリスナーの作成、登録の必要はありません。

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

4. 波形の取り込み開始

```
ScStart(handle);
```

5. ラッチ（波形を取得する場合は必ず必要です）
   測定情報（ヒストリ情報）をマークします。

```
ScLatchData(handle);
```

6. ヒストリ情報取得
   ラッチしたアクイジションカウントを読み出してヒストリが更新されているか確認をします。更新されていれば、データを読み出すアクイジションカウントを設定します。

```
ScGetLatchAcqCount(handle, &acqCount);
ScGetAcqCount(handle, acqCount);
```

7. 波形データ取得
   トリガによる波形取り込みの場合、ScGetAcqDataLength で取得できる点数は設定したレコード長になります。点数は、レコード長および T/Div（サンプルレート）に依存します。また、ScGetAcqData に渡すサイズはバイト数のため、あらかじめ ScGetChannelBits でデータの 1 点のサイズを求めておいてください。

```
char buff[100000];
ScGetAcqDataLength(handle, 1,0, &length);
ScGetAcqData(handle, 1, 0, buff, sizeof(buff), &count,
 &dataSize);
. . .
ScResumeAcquisition(handle);
```

   なお、1 回の ScGetAcqData で取得できるサイズは、通信仕様の制限により最大 999999999Byte です。そのため、レコード長が 500M 点以上の場合（電圧などのアナログモジュールの場合）、ScGetAcqData を複数回に分けて取得する必要があります。

   同期モードの場合、必要なチャネルのデータ取得が終了した時点で、アクイジションの再開（ScResumeAcquisition）を行います。

   波形の取り込み中は、5（ラッチ）から 7（波形データ取得）までを繰り返します。

8. 波形の取り込み停止

```
ScStop(handle);
```

9. 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);
```

10. 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（トリガモード）

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 ScSDKNet.dll を登録してください。名前空間は ScSDKNet、API は ScSDK クラス内のメソッドとして定義されています。

```
using ScSDKNet;
. . .
ScSDK api = new ScSDKNet.ScSDK();
api.ScInit();
```

2. 測定器（DL950/SL2000）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(ScSDK.SC_WIRE_USB, "91K225903",
 ScSDK.SC_TRIGGER, out handle);
```

3. イベントコールバックの登録
   トリガの同期モードの場合は、トリガエンドイベントを使用します。トリガエンドイベントを使用するには、コールバックメソッドを API に登録します。トリガエンド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。
   トリガの非同期モードの場合、イベントが発生しないためイベントリスナーの作成、登録の必要はありません。

```
private void trigEndCallback(int hndl, int type)
{
  . . .
}
api.ScAddCallback(hndl, trigEndCallback,
 ScSDK.SC_EVENTTYPE_TRIGEREND);
```

4. 波形の取り込み開始

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

5. ラッチ（波形を取得する場合は必ず必要です）
   測定情報（ヒストリ情報）をマークします。

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

6. ヒストリ情報の確認
   ラッチしたアクイジションカウントを読み出してヒストリが更新されているか確認をします。更新されていれば、データを読み出すアクイジションカウントを設定します。

```
api.ScGetLatchAcqCount(handle, out acqCount);
api.ScGetAcqCount(handle, out acqCount);
```

7. 波形データ取得
   トリガによる波形取り込みの場合、ScGetAcqDataLength で取得できる点数は設定したレコード長になります。（点数は、レコード長および T/Div（サンプルレート）に依存します。また、ScGetAcqData に渡すサイズはバイト数のため、あらかじめ ScGetChannelBits でデータの 1 点のサイズを求めておいてください。

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

   なお、1 回の ScGetAcqData で取得できるサイズは、通信仕様の制限により最大 999999999Byte です。そのため、レコード長が 500M 点以上の場合（電圧などのアナログモジュールの場合）、ScGetAcqData を複数回に分けて取得する必要があります。

   同期モードの場合、必要なチャネルのデータ取得が終了した時点、アクイジションの再開（ScResumeAcquisition）を行います。

   波形の取り込み中は、5（ラッチ）から 7（波形データ取得）までを繰り返します。

8. 波形の取り込み停止

```
api.ScStop(handle);
```

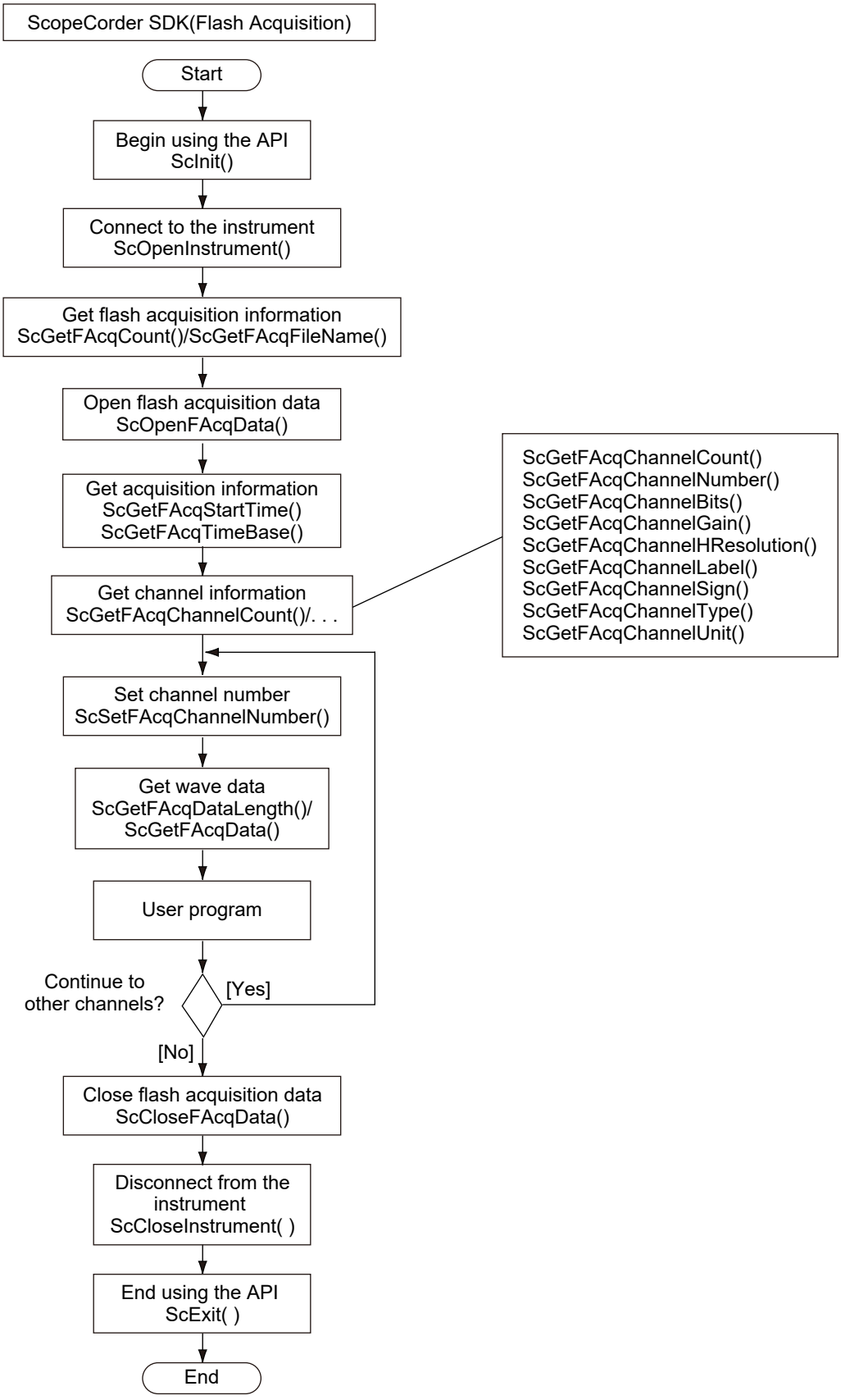
9. 測定器との通信を切断（必ず必要です）
   この API を呼び出した後、測定器ハンドルは無効になります。

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

10. API の使用終了（必ず必要です）

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

フラッシュアクイジションデータアクセスライブラリ機能



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 "ScSDK.h"
. . .
ScInit();
. . .
```

2. 測定器（DL950/SL2000）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. 波形データのリスト取得（オープンのために ScGetFAcqFileName は必要です）
   フラッシュアクイジションで記録した波形データのリストを取得します。

```
int facqCount;
char name[500][64];
ScGetFAcqCount(handle, &facqCount);
for(int i = 0; i < facqCount; i++){
  ScGetFAcqFileName(handle, i+1, &name[i][0]);
}
```

4. 波形データのオープン（必ず必要です）
   転送したい波形データをオープンします。ファイル名は 3 で取得したものを使用します。

```
ScOpenFAcqData(handle, name);
```

5. 波形データのチャネル数とチャネル番号の取得（波形データの取得には ScGetFAcqChannleNumber で取得したチャネル番号が必要です）
   オープンした波形データに含まれるチャネル数とチャネル番号を取得します。

```
int chCount;
int chNo[500];
int subChNo[500];
ScGetFAcqChannelCount(handle, &chCount);
for(int i = 0; i < chCount; i++){
  ScGetFAcqChannelNumber(handle, i+1, &chNo[i], &subChNo[i]);
}
```

6. 波形データに保存されている各チャネルの設定情報を取得
   5 で取得したチャネル番号を用いて保存されている各チャネルの設定情報を取得します。取得した設定情報は、各チャネルの波形データの物理値変換などに使用します。

```
char type[500][16];
char label[500][32];
int bits[500];
double gain[500];
double offset[500];
double hreso[500];
for(int i = 0; i < chCount; i++){
  ScGetFAcqChannelBits(handle, chNo[i], subChNo[i], &bits[i]);
  ScGetFAcqChannelGain(handle, chNo[i], subChNo[i], &gain[i]);
  ScGetFAcqChannelOffset(handle, chNo[i], subChNo[i],
   &offset[i]);
  ScGetFAcqChannelHResolution(handle, chNo[i], subChNo[i],
   &bits[i], &hreso[i]);
  ScGetFAcqChannelLabel(handle, chNo[i], subChNo[i],
   &label[i]);
  ScGetFAcqChannelType(handle, i, chNo[i], subChNo[i],
   &type[i]);
}
```

7. 各チャネルの波形データを取得
   5 で取得したチャネル番号を用いて保存されている各チャネルの波形データを取得します。

```
__int64 length;
char buff[1000000];
int rcv_len;
for(int i = 0; i < chCount; i++){
ScSetFAcqChannelNumber(handle, chNo[i], subChNo[i]);
ScGetFAcqDataLength(handle, &length);
while(1){
  ScGetFAcqData(handle, buff, sizeof(buff), &rcv_len);
  if(rcv_len == 0){
    // no more data
    break;
  }
  . . .
}
```

   上記の例にあるように rcv_len が 0 になるまで ScGetFAcqData を繰り返すことで保存されているチャネルのデータをすべて取得できます。なお一回に読み出す最大サイズは ScSetFAcqDataSize で指定できます。

8. 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);
```

9. 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 ScSDKNet.dll を登録してください。名前空間は ScSDKNet、API は ScSDK クラス内のメソッドとして定義されています。

```
using ScSDKNet;
. . .
ScSDK api = new ScSDKNet.ScSDK();
api.ScInit();
```

2. 測定器（DL950/SL2000）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(ScSDK.SC_WIRE_USB, "91K225903",
 out handle);
```

3. 波形データのリスト取得（オープンのために ScGetFAcqFileName は必要です）
   フラッシュアクイジションで記録した波形データのリストを取得します。

```
int facqCount;
string[] name = new string[500];
api.ScGetFAcqCount(handle, out facqCount);
for(int i = 0; i < facqCount; i++){
  api.ScGetFAcqFileName(handle, i+1, out name[i]);
}
```

4. 波形データのオープン（必ず必要です）
   転送したい波形データをオープンします。ファイル名は 3 で取得したものを使用します。

```
api.ScOpenFAcqData(handle, fileName);
```

5. 波形データのチャネル数とチャネル番号の取得（波形データの取得には ScGetFAcqChannleNumber で取得したチャネル番号が必要です）
   オープンした波形データに含まれるチャネル数とチャネル番号を取得します。

```
int chCount;
int[] chNo = new int[500];
int[] subChNo = new int[500];
api.ScGetFAcqChannelCount(handle, out chCount);
for(int i = 0; i < chCount; i++){
  api.ScGetFAcqChannelNumber(handle, i+1, out chNo[i],
   out subChNo[i]);
}
```

6. 波形データに保存されている各チャネルの設定情報を取得
   5 で取得したチャネル番号を用いて保存されている各チャネルの設定情報を取得します。取得した設定情報は、各チャネルの波形データの物理値変換などに使用します。

```
char[] type = new char[500];
char[] label = new char[500];
int[] bits = new int[500];
double[] gain = new double[500];
double[] offset = new double[500];
double[] hreso= new double[500];
for(int i = 0; i < chCount; i++){
  api.ScGetFAcqChannelBits(handle, chNo[i], subChNo[i],
   out bits[i]);
  api.ScGetFAcqChannelGain(handle, chNo[i], subChNo[i],
   out gain[i]);
  api.ScGetFAcqChannelOffset(handle, chNo[i], subChNo[i],
   out offset[i]);
  api.ScGetFAcqChannelHResolution(handle, chNo[i], subChNo[i],
   out bits[i], out hreso[i]);
  api.ScGetFAcqChannelLabel(handle, chNo[i], subChNo[i],
   out label[i]);
  api.ScGetFAcqChannelType(handle, i, chNo[i], subChNo[i],
   out type[i]);
}
```

6. 各チャネルの波形データを取得
   5 で取得したチャネル番号を用いて保存されている各チャネルの波形データを取得します。

```
__int64 length;
byte[] buff = new byte[1000000];
int rcv_len;
for(int i = 0; i < chCount; i++){
api.ScSetFAcqChannelNumber(handle, chNo[i], subChNo[i]);
api.ScGetFAcqDataLength(handle, &length);
while(1){
  api.ScGetFAcqData(handle, buff, buff.Length, out rcv_len);
  if(rcv_len == 0){
    // no more data
    break;
  }
    . . .
}
```

   上記の例にあるように rcv_len が 0 になるまで ScGetFAcqData を繰り返すことで保存されているチャネルのデータをすべて取得できます。なお一回に読み出す最大サイズは ScSetFAcqDataSize で指定できます。

8. 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);
```

9. API の使用終了（必ず必要です）

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

3.1　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:
    /*!
     * \brief Overrun handler
     * \param handle API handle

     * \param\ reserve
     */

    virtual void handleEventScCallListener(ScHandle handle, __int64 reserve){}
    virtual void handleEventScTrigEnd(ScHandle handle){}
};
```

詳細 :

登録可能なイベントはフリーランの「オーバーランイベント」およびトリガの同期モードにおける「トリガエンドイベント」となります。
handleEventScCallListener をオーバーライドすることによって、オーバーラン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。
handleEventScTrigEnd をオーバーライドすることによって、トリガエンド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。
ScAddEventListener でインスタンスを登録します。

3.2 固定値定義

SC_SUCCESS

概要 :

正常

書式 :

#define SC_SUCCESS

詳細 :

API の各関数の返却値定義

SC_ERROR

概要 :

エラー

書式 :

#define SC_ERROR

詳細 :

API の各関数の返却値定義

SC_UNOPENED

概要 :

ファイル未指定

書式 :

#define SC_UNOPENED

詳細 :

API の各関数の返却値定義（フラッシュアクイジションデータアクセスライブラリ機能）

SC_USE_ACQMEMORY

概要 :

ACQ メモリーに測定データが存在する

書式 :

#define SC_USE_ACQMEMORY

詳細 :

ScOpenFAcqData の返却値定義

SC_ERR_UNOPENED

概要 :

ファイル未指定の場合のエラー

書式 :

#define SC_ERR_UNOPENED

詳細 :

API の各関数の返却値定義（フラッシュアクイジションデータアクセスライブラリ機能）

SC_ERR_RUNNING

概要 :

接続を行った機器が測定中の場合のエラー

書式 :

#define SC_ERR_RUNNING

詳細 :

ScOpenInstrument 等の Open 系の関数を実行したときの返却値定義

SC_ERR_SYNC_CONN

概要 :

接続を行った機器が複数台同期接続中の場合のエラー

書式 :

#define SC_ERR_SYNC_CONN

詳細 :

ScOpenInstrument 等の Open 系の関数を実行したときの返却値定義

SC_ERR_SYNC_SUB

概要 :

複数台同期中のサブ機の場合のエラー

書式 :

#define SC_ERR_SYNC_SUB

詳細 :

ScOpenInstrument 等の Open 系の関数や、データアクイジション関連の関数を実行したときの返却値定義

SC_ERR_RECORDER

概要 :

接続を行った機器がレコーダモードだった場合のエラー

書式 :

#define SC_ERR_RECORDER

詳細 :

ScOpenInstrument 等の Open 系の関数を実行したときの返却値定義

SC_ERR_MODE

概要 :

再接続を行った際に指定したモードと機器の設定が異なる場合のエラー

書式 :

#define SC_ERR_MODE

詳細 :

ScReopenInstrument を実行したときの返却値定義

SC_ERR_NOTAPPLICABLE

概要 :

接続を行った機器が DL950/SL2000 シリーズ以外、もしくは対象外のバージョンだった場合のエラー

書式 :

#define SC_ERR_NOTAPPLICABLE

詳細 :

ScOpenInstrument 等の Open 系の関数を実行したときの返却値定義

SC_ERR_NODATA

概要 :

指定したフラッシュアクイジション番号に該当する波形データのファイル名がない、もしくは保存する波形がなかった場合のエラー

書式 :

#define SC_ERR_NODATA

詳細 :

ScGetFAcqFileName 及び ScSaveTriggerWDF を実行したときの返却値定義

SC_ERR_PARAMETER

概要 :

関数の引数不正の場合のエラー

書式 :

#define SC_ERR_PARAMETER

詳細 :

関数の引数が不正のときの返却値定義

SC_WIRE_USBTMC

概要 :

USB 回線（YTUSB）

書式 :

#define SC_WIRE_USBTMC

詳細 :

DL950/SL2000 シリーズに接続する時の回線種別定義
※ USB（TMCTL 標準ドライバ）接続時に選択します。

SC_WIRE_VISAUSB

概要 :

USB 回線（VISAUSB）

書式 :

#define SC_WIRE_VISAUSB

詳細 :

DL950/SL2000 シリーズに接続するときの回線種別定義
※ USB（VISA ドライバ使用時）接続時に選択します。

SC_WIRE_VXI11

概要 :

イーサネット回線（VXI11）

書式 :

#define SC_WIRE_VXI11

詳細 :

DL950/SL2000 シリーズに接続するときの回線種別定義
※ GigaBitEther での接続時に使用します。

SC_WIRE_HISLIP

概要 :

イーサネット回線（HiSLIP）

書式 :

#define SC_WIRE_HISLIP

詳細 :

DL950/SL2000 シリーズに接続するときの回線種別定義
※ 10G による高速データ転送モードを使用する場合に選択します。

SC_FREERUN

概要 :

フリーラン動作

書式 :

#define SC_FREERUN

詳細 :

フリーランによる波形取り込みを実施する場合に指定します。
DL950/SL2000 から受信するデータを、そのままプログラムにブロックデータとして渡します。

SC_TRIGGER

概要 :

トリガ同期モード

書式 :

#define SC_TRIGGER

詳細 :

トリガ同期モードで波形データを取得する場合に指定します。
API を使用して明示的に DL950/SL2000 の波形取り込みのシーケンスを制御します。

SC_TRIGGER_ASYNC

概要 :

トリガ非同期モード

書式 :

#define SC_TRIGGER_ASYNC

詳細 :

トリガ非同期モードで波形データを取得する場合に指定します。
データ取得の完了と関係なく DL950/SL2000 での波形取り込みが進みます。

SC_NOMODE

概要 :

接続時のモード設定をしない

書式 :

#define SC_NOMODE

詳細 :

測定中の機器に接続する、もしくはフラッシュアクイジションデータアクセスライブラリ機能を使用する場合に指定します。

SC_EVENTTYPE_OVERRUN

概要 :

イベントタイプ（オーバーラン）

書式 :

#define SC_EVENTTYPE_OVERRUN

詳細 :

フリーラン時のオーバーランのイベントコールバックを登録する際のイベントタイプを指定します。
.NET 版（C#）のみで使用します。

SC_EVENTTYPE_TRIGGEREND

概要 :

イベントタイプ（トリガエンド）

書式 :

#define SC_EVENTTYPE_TRIGEREND

詳細 :

トリガモード時のトリガエンドのイベントコールバックを登録する際のイベントタイプを指定します。
.NET 版（C#）のみで使用します。

SC_SIZE_16MB

概要 :

データ送信サイズを 16MiB に指定

書式 :

#define SC_SIZE_16MB

詳細 :

データ送信サイズの定義

SC_SIZE_32MB

概要 :

データ送信サイズを 32MiB に指定

書式 :

#define SC_SIZE_32MB

詳細 :

データ送信サイズの定義

SC_SIZE_64MB

概要 :

データ送信サイズを 64MiB に指定

書式 :

#define SC_SIZE_64MB

詳細 :

データ送信サイズの定義

SC_SIZE_128MB

概要 :

データ送信サイズを 128MiB に指定

書式 :

#define SC_SIZE_128MB

詳細 :

データ送信サイズの定義

SC_SIZE_256MB

概要 :

データ送信サイズを 256MiB に指定

書式 :

#define SC_SIZE_256MB

詳細 :

データ送信サイズの定義

SC_SIZE_512MB

概要 :

データ送信サイズを 512MiB に指定

書式 :

#define SC_SIZE_512MB

詳細 :

データ送信サイズの定義

SC_10GMODE_ON

概要 :

10Gbps 高速転送モードを有効にする

書式 :

#define SC_10GMODE_ON

詳細 :

10Gbps 高速転送モードの設定の際に使用します。

SC_10GMODE_OFF

概要 :

10Gbps 高速転送モードを無効にする

書式 :

#define SC_10GMODE_OFF

詳細 :

10Gbps 高速転送モードの設定の際に使用します。

SC_DRIVE_IDRIVE

概要 :

カレントドライブを IDRive に指定する。

書式 :

#define SC_DRIVE_IDRIVE

詳細 :

カレントドライブの設定の際に使用します。

SC_DRIVE_NETWORK

概要 :

カレントドライブを NETWork に指定する。

書式 :

#define SC_DRIVE_NETWORK

詳細 :

カレントドライブの設定の際に使用します。

SC_DRIVE_SD

概要 :

カレントドライブを SD に指定する。

書式 :

#define SC_DRIVE_SD

詳細 :

カレントドライブの設定の際に使用します。

SC_DRIVE_USB_0

概要 :

カレントドライブを USB-0 に指定する。

書式 :

#define SC_DRIVE_USB_0

詳細 :

カレントドライブの設定の際に使用します。

SC_DRIVE_USB_1

概要 :

カレントドライブを USB-1 に指定する。

書式 :

#define SC_DRIVE_USB_1

詳細 :

カレントドライブの設定の際に使用します。

SC_DRIVE_FLASH

概要 :

カレントドライブを FLASh に指定する。

書式 :

#define SC_DRIVE_FLASH

詳細 :

カレントドライブの設定の際に使用します。

SC_FILE_ETE_ALL

概要 :

取得するファイル拡張子をすべてに指定する。

書式 :

#define SC_FILE_ETE_ALL

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_SET

概要 :

取得するファイル拡張子を *.SET に指定する。

書式 :

#define SC_FILE_ETE_SET

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_WDF

概要 :

取得するファイル拡張子を *.WDF に指定する。

書式 :

#define SC_FILE_ETE_WDF

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_BMP

概要 :

取得するファイル拡張子を *.BMP に指定する。

書式 :

#define SC_FILE_ETE_BMP

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_PNG

概要 :

取得するファイル拡張子を *.PNG に指定する。

書式 :

#define SC_FILE_ETE_PNG

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_JPG

概要 :

取得するファイル拡張子を *.JPG に指定する。

書式 :

#define SC_FILE_ETE_JPG

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_SNP

概要 :

取得するファイル拡張子を *.SNP に指定する。

書式 :

#define SC_FILE_ETE_SNP

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_SBL

概要 :

取得するファイル拡張子を *.SBL に指定する。

書式 :

#define SC_FILE_ETE_SBL

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_CSV

概要 :

取得するファイル拡張子を *.CSV に指定する。

書式 :

#define SC_FILE_ETE_CSV

詳細 :

ファイルリストの取得の際に使用します。

SC_FILE_ETE_MAT

概要 :

取得するファイル拡張子を *.MAT に指定する。

書式 :

#define SC_FILE_ETE_MAT

詳細 :

ファイルリストの取得の際に使用します。

SC_SYNC_OFF

概要 :

機器が複数台同期機能を使用していない状態であることを示す。

書式 :

#define SC_SYNC_OFF

詳細 :

ハンドルリスト、デバイスリスト及びステータス情報取得の際に使用します。

SC_SYNC_CONN

概要 :

機器が複数台同期機能の接続待機状態であることを示す。

書式 :

#define SC_SYNC_CONN

詳細 :

デバイスリスト及びステータス情報取得の際に使用します。

SC_SYNC_MAIN

概要 :

機器がメインユニットとして動作中であることを示す。

書式 :

#define SC_SYNC_MAIN

詳細 :

ハンドルリスト、デバイスリスト及びステータス情報取得の際に使用します。

SC_SYNC_SUB

概要 :

機器がサブユニットとして動作中であることを示す。

書式 :

#define SC_SYNC_SUB

詳細 :

ハンドルリスト、デバイスリスト及びステータス情報取得の際に使用します。

SC_STAT_STOPPED

概要 :

機器が測定停止中であることを示す。

書式 :

#define SC_STAT_STOPPED

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_RUNNING

概要 :

機器が測定中であることを示す。

書式 :

#define SC_STAT_RUNNING

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_INTERNAL

概要 :

機器のタイムベースが内部であることを示す。

書式 :

#define SC_ STAT_INTERNAL

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_EXTERNAL

概要 :

機器のタイムベースが外部であることを示す。

書式 :

#define SC_STAT_EXTERNAL

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_SSD

概要 :

機器のリアルタイム記録が SSD 記録であることを示す。

書式 :

#define SC_STAT_SSD

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_FACQ

概要 :

機器のリアルタイム記録がフラッシュアクイジションであることを示す。

書式 :

#define SC_STAT_FACQ

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_TRIGGER

概要 :

機器のアクイジションモードがトリガであることを示す。

書式 :

#define SC_STAT_TRIGGER

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_FREERUN

概要 :

機器のアクイジションモードがフリーランであることを示す。

書式 :

#define SC_STAT_FREERUN

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_OFF

概要 :

機器の状態設定が OFF であることを示す。

書式 :

#define SC_STAT_OFF

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_ON

概要 :

機器の状態設定が ON であることを示す。

書式 :

#define SC_STAT_ON

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_ST1

概要 :

機器の内蔵ストレージオプションが ST1 であることを示す。

書式 :

#define SC_STAT_ST1

詳細 :

ステータス情報取得の際に使用します。

SC_STAT_ST2

概要 :

機器の内蔵ストレージオプションが ST2 であることを示す。

書式 :

#define SC_STAT_ST2

詳細 :

ステータス情報取得の際に使用します。

SC_SORT_NAME_ASC

概要 :

ファイルリストをファイル名の昇順に取得します。

書式 :

#define SC_SORT_NAME_ASC

詳細 :

ファイルリストを取得する際に使用します。

SC_SORT_NAME_DESC

概要 :

ファイルリストをファイル名の降順に取得します。

書式 :

#define SC_SORT_NAME_DESC

詳細 :

ファイルリストを取得する際に使用します。

SC_SORT_DATE_ASC

概要 :

ファイルリストをファイル更新日時の昇順に取得します。

書式 :

#define SC_SORT_DATE_ASC

詳細 :

ファイルリストを取得する際に使用します。

SC_SORT_DATE_DESC

概要 :

ファイルリストをファイル更新日時の降順に取得します。

書式 :

#define SC_SORT_DATE_DESC

詳細 :

ファイルリストを取得する際に使用します。

SC_SORT_SIZE_ASC

概要 :

ファイルリストをファイルサイズの昇順に取得します。

書式 :

#define SC_SORT_SIZE_ASC

詳細 :

ファイルリストを取得する際に使用します。

SC_SORT_SIZE_DESC

概要 :

ファイルリストをファイルサイズの降順に取得します。

書式 :

#define SC_SORT_SIZE_DESC

詳細 :

ファイルリストを取得する際に使用します。

3.3 データ構造定義

HandleList

概要 :

ハンドルリスト

書式 :

```
struct HandleList {
    ScHandle    handle;         // 接続機器のハンドル
    int         sync;           // 接続機器の複数台同期の状態
    char        unit[32];       // ユニット名
};
```

詳細 :

ハンドルリストを取得する際に使用します。
各パラメータには以下の固定値が保存されます。
Sync　　SC_SYNC_OFF / SC_SYNC_MAIN / SC_SYNC_SUB

DeviceList

概要 :

デバイスリスト

書式 :

```
struct DeviceList {
    char        adr[64];        // アドレス
    int         sync;           // 接続機器の複数台同期の状態
    char        name[32];       // 機器名
    char        unit[32];       // ユニット名
};
```

詳細 :

デバイスリストを取得する際に使用します。
各パラメータには以下の固定値が保存されます。
Sync　　SC_SYNC_OFF / SC_SYNC_CONN / SC_SYNC_MAIN / SC_SYNC_SUB

StatusInfo

概要 :

ステータス情報

書式 :

```
struct StatusInfo {
    int         Running;        // 測定状態
    int         Sync;           // 複数台同期接続設定
    int         TimeBase;       // タイムベース
    int         RealTime;       // リアルタイム記録設定
    int         AcqMode;        // アクイジションモード
    int         DualCapture;    // デュアルキャプチャ設定
    int         GONogo;         // GONogo 設定
    int         StorageOpt;     // ストレージオプション
};
```

詳細 :

デバイスの設定を取得する際に使用します。
各パラメータには以下の固定値が保存されます。

Running	SC_STAT_STOPPED / SC_STAT_RUNNING
Sync	SC_SYNC_OFF / SC_SYNC_CONN / SC_SYNC_MAIN / SC_SYNC_SUB
SystemMode	SC_STAT_SCOPE / SC_STAT_RECODER
TimeBase	SC_STAT_INTERNAL / SC_STAT_EXTERNAL
RealTime	SC_STAT_OFF / SC_STAT_SSD / SC_STAT_FACQ
AcqMode	SC_STAT_TRIGGER / SC_STAT_FREERUN
DualCapture	SC_STAT_OFF / SC_STAT_ON
GONogo	SC_STAT_OFF / SC_STAT_ON
StorageOpt	SC_STAT_OFF / SC_STAT_ST1 / SC_STAT_ST2

FileInfo

概要 :

ファイル情報

書式 :

```
struct FileInfo {
    char          name[256];      // ファイル名
    unsigned int  size;           // ファイルサイズ
    char          date[10];       // ファイル保存日時
    char          time[5];        // ファイル保存時刻
    char          rw[5];          // リードライト
};
```

詳細 :

ファイル情報及びファイル情報リストを取得する際に使用します。

4.1　共通 API

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 "ScSDK.h"
...
if (ScInit() == SC_SUCCESS) {
  ...
}
```

使用例 :[C#]

```
using ScSDKNet;
...
ScSDKNet.ScSDK api = new ScSDKNet.ScSDK();
if (api.ScInit() == ScSDK.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, int mode, ScHandle* rHndl);
[C#]
int ScOpenInstrument(int wire, string address, int mode, out int rHndl);

引数 :

[IN] wire	回線種別	
	SC_WIRE_USBTMC	USBTMC（YTUSB）
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	接続先アドレス（USB の場合は測定器シリアル番号）	
[IN] mode	接続モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード
	SC_NOMDOE	接続モードなし
[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
SC_ERR_RUNNING	測定中エラー
SC_ERR_SYNC_CONN	複数台同期接続中エラー
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー
SC_ERR_RECORDER	レコーダモードエラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

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、mode によって自動的に測定器の波形の取り込み条件を設定します。
SC_NOMODE を選択した場合、現在設定されている接続モードを維持します。
SC_NOMODE 以外の mode を選択した場合、機器が測定中の場合はエラーとします。

注意 :

ひとつの測定器を多重に接続することはできません。
10Gbps イーサネットを使用する場合は、SC_WIRE_HISLIP を選択してください。
DL950/SL2000 がレコーダモードのときには接続できません。

使用例 :[C++]

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

使用例 :[C#]

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

ScReopenInstrument

概要 :

測定器に接続する（測定中の機器も可）

書式 :

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

引数 :

[IN] wire	回線種別	
	SC_WIRE_USBTMC	USBTMC（YTUSB）
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	接続先アドレス（USB の場合は測定器シリアル番号）	
[IN] mode	接続モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード
[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
SC_ERR_SYNC_CONN	複数台同期接続中エラー
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー
SC_ERR_RECORDER	レコーダモードエラー
SC_ERR_MODE	モード違いエラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

測定中の測定器に接続し、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。
測定を停止しているときは、指定した mode を測定器へ自動的に設定します。

注意 :

ひとつの測定器を多重に接続することはできません。
10Gbps イーサネットを使用する場合は、SC_WIRE_HISLIP を選択してください。
DL950/SL2000 がレコーダモードのときには、接続できません。
測定中で、指定した mode と測定器の状態が異なる場合は、エラーとします。

使用例 :[C++]

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

使用例 :[C#]

```
int hndl;
if (api.ScReopenInstrument(ScSDK.SC_WIRE_USB, "91K225895" ,
 ScSDK.SC_FREERUN, out hndl) == ScSDK.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] hndl	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 :

ScOpenInstrument で接続した測定器から切断します。
測定器が測定中の場合切断に時間がかかりますが、ScSetTriggerTimeout でタイムアウト時間を短くすることによって短時間で切断することができます。

注意 :

この API を呼び出した後、ハンドルは無効になります。

ScOpenInstrumentEx

概要 :

複数台同期中の測定器に接続する

書式 :

[C++]
ScResult ScOpenInstrumentEx(int wire, char* address, int mode, HandleList* List, int max,
 int* ListCount);
[C#]
int ScOpenInstrumentEx(int wire, string address, int mode, out HandleList[] List, int max,
 out int ListCount);

引数 :

[IN] wire	回線種別	
	SC_WIRE_USBTMC	USBTMC（YTUSB）
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	Main 機の接続先アドレス （USB の場合は測定器シリアル番号）	
[IN] mode	接続モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード
	SC_NOMDOE	接続モードなし
[OUT] List	測定器ハンドルリスト	
[IN] max	測定器ハンドルリストの最大サイズ	
[OUT] ListCount	測定器ハンドルリスト数	

戻り値 :

SC_SUCCESS	接続成功
SC_ERROR	接続エラー
SC_ERR_RUNNING	測定中エラー
SC_ERR_SYNC_CONN	複数台同期接続中エラー
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー
SC_ERR_RECORDER	レコーダモードエラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

複数台同期中の Main 機のアドレスから Sub 機に接続し、各測定器のハンドルを取得します。
TMCTL ライブラリの API で取得した測定器ハンドルを使用して測定器の本 API の接続処理を実行します。
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、mode によって自動的に測定器の波形の取り込み条件を設定します。
max は配列の最大サイズを指定します。

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。
10Gbps イーサネットを使用する場合は、SC_WIRE_HISLIP を選択してください。
DL950/SL2000 がレコーダモードのときには接続できません。
本 API で使用する測定器ハンドルは、TMCTL ライブラリの API で接続した DL950/SL2000 のみを想定しています。それ以外で取得した測定器ハンドルでは正しく動作しません。

ScReopenInstrumentEx

概要 :

複数台同期中の測定器に接続する（測定中の機器も可）

書式 :

[C++]

```
ScResult ScReopenInstrumentEx(int wire, char* address, int mode, HandleList* List,
 int max, int* ListCount);
```

[C#]

```
int ScReopenInstrumentEx(int wire, string address, int mode, out HandleList[] List,
 int max, out int ListCount);
```

引数 :

[IN] wire	回線種別	
	SC_WIRE_USBTMC	USBTMC（YTUSB）
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	接続先アドレス（USB の場合は測定器シリアル番号）	
[IN] mode	接続モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード
[OUT] List	測定器ハンドルリスト	
[IN] max	測定器ハンドルリストの最大サイズ	
[OUT] ListCount	測定器ハンドルリスト数	

戻り値 :

SC_SUCCESS	接続成功
SC_ERROR	接続エラー
SC_ERR_SYNC_CONN	複数台同期接続中エラー
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー
SC_ERR_RECORDER	レコーダモードエラー
SC_ERR_MODE	モード違いエラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

測定中の測定器に接続し、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。
測定を停止しているときは、指定した mode を測定器へ自動的に設定します。

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。
10Gbps イーサネットを使用する場合は、SC_WIRE_HISLIP を選択してください。
DL950/SL2000 がレコーダモードのときには接続できません。
測定中で、指定した mode と測定器の状態が異なる場合は、エラーとします。

ScCloseInstrumentEx

概要 :

複数台同期中の測定器から切断する

書式 :

[C++]
ScResult ScCloseInstrumentEx(HandleList* List, int ListCount);
[C#]
int ScCloseInstrumentEx(HandleList[] List, int ListCount);

引数 :

[IN] List	測定器ハンドルリスト
[IN] ListCount	測定器ハンドルリスト数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ScOpenInstrumentEx で接続した測定器から切断します。

注意 :

この API を呼び出した後、本 API での測定器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ー
[IN] buffLen	バッファーのサイズ
[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) == ScSDK.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, int* endFlg);
[C#]
int ScGetBinaryData<DT>(int hndl, string command, ref DT[] buff, int buffLen,
out int receiveLen, out endFlg);

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 受信中のデータを受信する場合は 0（ヌルポインタ）を指定	
[OUT] buff	バイナリデータを受信するバッファー	
[IN] buffLen	バイナリデータを受信するバッファーのサイズ（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）	
[OUT] endFlg	受信終了フラグ	
	0	受信中（残データあり）
	1	受信終了

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。
buffLen で指定したバッファーサイズが、実際に受信したバイナリデータのサイズよりも小さい場合に、endFlg が 0 になります。
ScGetBinaryData、ScGetLatchAcqData、ScGetAcqData の受信終了フラグが 1 ではない場合にバイナリデータを継続して受信する場合は、command に 0（ヌルポインタ）を指定してください。

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) == ScSDK.SC_SUCCESS) {
    ...
  }
```

ScQueryMessage

概要 :

通信コマンドを発行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	受信バッファー
[IN] buffLen	受信バッファーの長さ（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、buffLen に受信バッファーの長さではなく、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) == ScSDK.SC_SUCCESS)
{
  ...
}
```

ScSet10GMode

概要 :

10Gbps 高速データ転送モードを設定する

書式 :

[C++]
ScResult ScSet10GMode(ScHandle hndl, int onoff);
[C#]
int ScSet10GMode(int hndl, int onoff);

引数 :

[IN] hndl	測定器ハンドル	
[IN] onoff	10Gbps 高速データ転送モードの設定	
	0	10Gbps 高速データ転送モード無効
	1	10Gbps 高速データ転送モード有効

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

データアクイジションまたはフラッシュアクイジションのデータ転送を行う際に、ハードウェアを利用した 10Gbps による高速データ転送を行うかどうかを設定します。10G オプションが実装されている場合に使用できます。

注意 :

本コマンドは 10Gbps イーサネットによる接続がされていて、回線種別に HiSLIP が選択されている場合に、有効となります。
本コマンドは、波形の取り込み開始（ScStart）を実行する前に行うようにしてください。波形の取り込み中には変更できません
本コマンドは ScGetFAcqData を実行する前に行ってください。転送の途中に設定を変更すると転送が正常に行えなくなる場合があります。その場合、クローズを実行して再度やり直してください。
本モードが無効でも 10Gbps イーサネットによるデータ転送は可能ですが、転送パフォーマンスが落ちるためデータアクイジションで使用する場合、オーバーランが発生しやすくなります。
ファイル転送では本機能は無効となり、ソフトウェアによる転送となります。10Gbps イーサネット接続時でも転送速度は 1Gbps イーサネット接続よりも遅くなる場合があります。

ScGet10GMode

概要 :

10Gbps 高速データ転送モードの設定値を確認する。

書式 :

[C++]
ScResult ScGet10GMode(ScHandle hndl, int *onoff);
[C#]
int ScGet10GMode(int hndl, out int onoff);

引数 :

[IN] hndl	測定器ハンドル	
[OUT] onoff	10Gps 高速データ転送モードの設定	
	0	10Gps 高速データ転送モード無効
	1	10Gps 高速データ転送モード有効

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

データ転送を行う際に、ハードウェアを利用した 10Gbps による高速データ転送モードが有効かどうかを確認します。

ScSearchDevices

概要 :

指定した回線につながっている測定器のリストを取得する

書式 :

[C++]
ScResult ScSearchDevices(int wire, DeviceList* list, int max, int* deviceCount,
char* option);
[C#]
int ScSearchDevices(int wire, out DeviceList[] list, int max, out int deviceCount,
string option);

引数 :

[IN] wire	回線種別	
	SC_WIRE_USBTMC	USBTMC（YTUSB）
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
[OUT] list	測定器のリスト	
[IN] max	測定器のリストの最大サイズ	
[OUT] deviceCount	測定器のリストの数	
[IN] option	ネットワークオプション文字列	

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_NOTAPPLICABLE	対象回線種別エラー
SC_ERR_PARAMETER	測定器のリストの最大サイズの値エラー

詳細 :

取得できる測定器のリストの最大数は 128 です。
max は配列の最大サイズを指定します。
ファームウェアのバージョンが Ver2.01 以降の DL950/SL2000 で使用できます。

ScGetStatusInfo

概要 :

接続先のステータス情報の取得する

書式 :

[C++]
ScResult ScGetStatusInfo(ScHandle hndl, StatusInfo *statusInfo);
[C#]
int ScGetStatusInfo(int hndl, out StatusInfo statusInfo);

引数 :

[IN] hndl	測定器ハンドル
[IN] statusInfo	接続先のステータス情報

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

接続機器のステータス情報を取得できます。

ScTmcSetTimeout

概要 :

TMCTL のタイムアウト時間を設定する

書式 :

[C++]
ScResult ScTmcSetTimeout(ScHandle hndl, int timeout);
[C#]
int ScTmcSetTimeout(ScHandle hndl, int timeout);

引数 :

[IN] hndl	測定器ハンドル
[IN] timeout	TMCTL のタイムアウト時間（100 ミリ秒単位）（0 ～ 65536）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

デフォルトのタイムアウト時間は 30 秒です。
0 の場合は、タイムアウト時間はなしに設定されます。

4.2 データアクイジション機能用 API

ScSetMeasuringMode

概要 :

測定モードを設定する

書式 :

[C++]
ScResult ScSetMeasuringMode(ScHandle hndl, int mode);
[C#]
int ScSetMeasuringMode(int hndl, int mode);

引数 :

[IN] hndl	測定器ハンドル	
[IN] mode	測定モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_RUNNING	測定中エラー
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー

詳細 :

測定モードを設定します。本 API は停止中のみ有効です。

ScSetMeasuringModeEx

概要 :

複数台同期中の測定モードを設定する

書式 :

[C++]
ScResult ScSetMeasuringModeEx(HandleList* List, int num, int mode)
[C#]
int ScSetMeasuringModeEx(HandleList[] List, int num, int mode)

引数 :

[IN] List	測定器ハンドルリスト	
[IN] num	測定器ハンドルリスト数	
[IN] mode	測定モード	
	SC_FREERUN	フリーラン
	SC_TRIGGER	トリガ同期モード
	SC_TRIGGER_ASYNC	トリガ非同期モード

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_RUNNING	測定中エラー

詳細 :

測定モードを設定します。本 API は停止中のみ有効です。

ScStart

概要 :

波形の取り込みを開始する

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
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー

詳細 :

波形の取り込みを開始します。(「Start」コマンドを送信します)

ScStartEx

概要 :

複数台同期中の測定器の波形の取り込みを開始する

書式 :

[C++]
ScResult ScStartEx(HandleList* List, int num);
[C#]
int ScStartEx(HandleList[] List, int num);

引数 :

[IN] List	測定器ハンドルリスト
[IN] num	測定器ハンドルリスト数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

波形の取り込みを開始します。(「Start」コマンドを送信します)

ScStop

概要 :

波形の取り込みを停止する

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
SC_ERR_SYNC_SUB	複数台同期サブ機接続エラー

詳細 :

波形の取り込みを停止します。(「Stop」コマンドを送信します)

ScStopEx

概要 :

複数台同期中の測定器の波形の取り込みを停止する

書式 :

[C++]
ScResult ScStopEx(HandleList* List, int num);
[C#]
int ScStopEx(HandleList[] List, int num);

引数 :

[IN] List	測定器ハンドルリスト
[IN] num	測定器ハンドルリスト数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

波形の取り込みを停止します。(「Stop」コマンドを送信します)

ScLatchData

概要 :

波形データをラッチする

書式 :

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

測定器内の波形データの現在の取り込み位置をマーキングします。
フリーランモードの場合、波形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。
トリガモードの場合、ヒストリ情報(アクイジション情報)に関する情報もマーキングします。

ScLatchDataEx

概要 :

複数台同期中の測定器の波形データをラッチする

書式 :

[C++]
ScResult ScLatchDataEx(HandleList* List, int num);
[C#]
int ScLatchDataEx(HandleList[] List, int num);

引数 :

[IN] List	測定器ハンドルリスト
[IN] num	測定器ハンドルリスト数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

測定器内の波形データの現在の取り込み位置をマーキングします。
フリーランモードの場合、波形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。
トリガモードの場合、ヒストリ情報（アクイジション情報）に関する情報もマーキングします。

ScGetLatchRawData

概要 :

フリーランモードでラッチ済みの波形データを取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] buff	データを保存するバッファー
[IN] buffLen	データを保存するバッファーの長さ
[OUT] receiveLen	受信したバイナリデータのサイズ（バイト数）
[OUT] endFlg	受信終了フラグ
	0　　受信中（残データあり）
	1　　受信終了

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ラッチ済みの波形データを取得します。
buffLen で指定したバッファーサイズが、実際に受信したバイナリデータのサイズよりも小さい場合に、endFlg が 0 になります。

注意 :

波形データは有効な測定チャネルがすべて含まれるブロック形式のデータとなります。
ブロック形式の詳細については、5.1 節の「ScGetLatchRawData のデータ構造について」を参照してください。
返却される波形データは AD 値です。
物理値に変換するには、ScGetChannelType で得られるデータ種別に応じた変換処理が必要です。以下の計算式で求めます。
物理値＝ AD 値× Gain+Offset
（Gain は ScGetChannelGain、Offset は ScGetChannelOffset で取得できます）
バッファーの長さは、「必要なメモリーサイズについて」を確認し、十分な長さとなるように設定してください。
10G による高速データ
endFlag が 0 の場合、残りのデータは ScGetBinaryData を使用して受信できます。
SC_FREERUN モード指定時に使用できます。

使用例 :[C++]

```
char buff[100000];
int size;
int endFlg;
if (ScGetLatchRawData(hndl,  buff, sizeof(buff), &size, &endFlg)
 == SC_SUCCESS) {
  ...
}
```

使用例 :[C#]

```
byte[] buff = new byte[100000];
int size;
int endFlg;
if (api.ScGetLatchRawData<byte>(hndl, ref buff, buff.Length,
 out size, out endFlg) == ScSDK.SC_SUCCESS)
{
   ...
}
```

ScGetChAcqData

概要 :

ScGetLatchRawData で取得したデータから指定チャネルの波形データの位置を取得する

書式 :

[C++]

ScResult ScGetChAcqData(int chNo, int subChNo, char* buff, int length, int* chOffset, int* chSize, unsigned int* timeSec, unsigned int* timeTick);

[C#]

int ScGetChAcqData<DT>(int chNo, int subChNo, DT[] buff,int length, out int chOffset, out int chSize, out unsigned int timeSec, out unsigned int timeTick);

引数 :

[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）
[IN] buff	ブロック形式のデータが保存されているバッファー
[IN] length	ブロック形式のデータが保存されているバッファーの長さ
[OUT] chOffset	チャネルデータの先頭へのオフセット位置（バイト数）
[OUT] chSize	チャネルのデータサイズ（バイト数）
[OUT] timeSec	取得したデータの先頭の時刻（UnixTime）
[OUT] timeTikc	取得したデータの先頭の時刻（ナノ秒）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

取得した波形データ（ブロック形式）から、指定チャネルのデータの位置を取得します。また取得したデータの先頭時刻を取得します。

プログラミング時の Tips:

サンプリングレートが高い波形の取り込みで、データの取りこぼし（オーバーランといいます）をしないようにするために、この ScGetChAcqData を使ってチャネルデータを取り出す場合、ScGetLatchRawData とは別スレッドにして取得したデータを解析することをお勧めします。

また、10G による高速ストリーミングによる波形の取り込みを行うような場合は、データの取りこぼし（オーバーランといいます）をしないようにするために、波形の取り込み中にはこの ScGetChAcqData を使用しないで ScGetLatchRawData で波形取得のみを行い、波形の取り込み終了後に改めて ScGetChAcqData によるチャネルデータの取り出しを行うような実装することをお勧めします。

注意 :

チャネルデータを保存するバッファーは、ScGetChannelBits を使用して 1 点のデータサイズとラッチ間の間隔から求められる十分なサイズを用意してください。
波形データは AD 値のため、物理値に変換するには、ScGetChannelType で得られるデータ種別に応じた変換処理が必要です。以下の計算式で求めます。
物理値＝ AD 値× Gain+Offset
（Gain は ScGetChannelGain、Offset は ScGetChannelOffset で取得できます）
指定されたチャネルのデータ存在しない場合は、エラーとなります。
ラッチ間の該当チャネルのデータが存在しないような場合は、データサイズが 0 となります。
ブロック形式の詳細については、5.1 節の「ScGetLatchRawData のデータ構造について」を参照してください。
SC_FREERUN モード指定時に使用できます。

使用例 :[C++]

```
char buff[100000];
int size;
if (ScGetLatchRawData(hndl,  buff, sizeof(buff), &size)
 == SC_SUCCESS) {
  int chOffset;
  int chSize;
  unsigned int timeSec,timeTick;
    if (ScGetChAcqData(1, 0, buff, size, &chOffset, &chSize,
     &timeSec, &timeTick) == SC_SUCCESS) {
       ...
    }
  ...
  }
```

使用例 :[C#]

```
byte[] buff = new byte[100000];
int size;
if (api.ScGetLatchRawData<byte>(hndl, buff, buff.Length,
 out size) == ScSDK.SC_SUCCESS) {
  int chOffset;
  int chSize;
  unsigned int timeSec;
  unsigned int timeTick;
  if (api.ScGetChAcqData<byte>(1, 0, buff, size, out chOffset,
   out chSize, out timeSec, out timeTick) == ScSDK.SC_SUCCESS) {
       ...
    }
  ...
  }
```

ScGetAcqData

概要 :

トリガモードでラッチ済みの波形データを取得する

書式 :

[C++]
ScResult ScGetAcqData(ScHandle hndl, int chNo, int subChNo, char* buff, int buffLen,
int* receiveLen, int* endFlg, unsigned int* timeSec, unsigned int* timeTick);
[C#]
int ScGetAcqData<DT>(int hndl, int chNo, int subChNo, ref DT[] buff, int buffLen,
out int recieveLen, out int endFlg, out unsigned int timeSec, out unsigned int timeTick);

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ー
[IN] buffLen	データを保存するバッファーの長さ
[OUT] receiveLen	取得したデータの長さ（バイト数）
[OUT] endFlg	受信終了フラグ
	0　受信中（残データあり）
	1　受信終了
[OUT] timeSec	取得したデータの先頭の時刻（UnixTime）
[OUT] timeTikc	取得したデータの先頭の時刻（ナノ秒）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

ラッチ済みの波形データを取得します。
また波形データの先頭時刻を取得します。
本 API を使用してトリガモードでの波形取り込み行う場合の動作概要については、5.1 節の「トリガモードについて」を参照してください
buffLen で指定したバッファーサイズが、実際に受信したバイナリデータのサイズよりも小さい場合に、endFlg が 0 になります。この場合、残りのデータは ScGetBinaryData を使用して受信できます。
外部サンプルでの波形取り込みを行った場合、timeSec、timeTick には時刻情報ではなくサンプルカウントの値が保存されます。詳細は、5.1 節の「トリガモードについて」を参照してください。

注意 :

直前に ScGetAcqDataLength の呼び出しが必要です。
通信の仕様上、一度に転送できるバイナリデータは 999999999 バイトまでになっているため、設定レコード長によっては本 API を複数回に分けて実行する必要があります。
なお DL950/SL2000 本体から送られてくるバイナリデータは、1 回の転送で最大 999999872 バイト（取得チャネルが電圧モジュールの場合は 499999936 点分、RMath の場合は 249999968 点分）までとなります。（※通信回線上には付帯情報（32 バイト分）が付加されるため 999999872 バイトより大きくなります）。
取得する波形データは AD 値です。
物理値に変換するには、ScGetChannelType で得られるデータ種別に応じた変換処理が必要です。
SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

使用例 :[C++]

```
char buff[100000];
int size,endFlg;
unsigned int timeSec,timeTick;
if (ScGetAcqData(hndl, 1, 0, buff, sizeof(buff),
 &size, &endFlg, &timeSec, &timeTick) == SC_SUCCESS) {
  ...
}
```

使用例 :[C#]

```
byte[] buff = new byte[100000];
int size,endFlg;
unsigned int timeSec,timeTick;
if (api.ScGetAcqData<byte>(hndl, 1, 0, ref buff,
 buff.Length, out size, out endFlg, out timeSec, out timeTick)
 == ScSDK.SC_SUCCESS) {
  ...
}
```

ScGetAcqDataLength

概要 :

トリガモードでラッチ済み波形データのデータ点数を取得する

書式 :

[C++]
ScResult ScGetAcqDataLength(ScHandle hndl, int chNo, int subChNo, __int64* length);
[C#]
int ScGetAcqDataLength(int hndl, int chNo, int subChNo, out long length);

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] length	データ点数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_MODE	モード違いエラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

ScSetAcqCount で設定されているアクイジション番号の指定チャネルのデータ点数を取得できます。

注意 :

この API で取得できるのはデータ点数になります。
ScGetAcqData で使用するバッファーサイズは、バイト数で指定します。以下の計算式で求めます。
バッファーサイズ=チャネルデータのビット長×データ点数
（チャネルデータのビット長は ScGetChannelBits、データ点数は ScGetAcqDataLength で取得できます）
ScGetAcqData の直前に呼び出しが必要です。

SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

ScGetFreeRunDataLength

概要 :

フリーランモードでタイムベース基準の波形データの点数を取得する

書式 :

[C++]
ScResult ScGetFreeRunDataLength(ScHandle hndl, __int64* length);
[C#]
int ScGetFreeRunDataLength(int hndl, out long length);

引数 :

[IN] hndl	測定器ハンドル
[OUT] length	有効点数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_MODE	モード違いエラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

ScSaveFreeRunWDF で取得可能なフリーランの波形データの最大点数を取得します。

注意 :

SC_FREERUN モード指定時に使用できます。

ScGetLatchAcqCount

概要 :

トリガモードでラッチ済み最大アクイジション番号を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] count	ラッチ時点の最大アクイジション番号

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ラッチ時点での最大アクイジション番号を取得します。
取得した値は ScSetAcqCount で使用します。

注意 :

SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

ScGetAcqCount

概要 :

トリガモードでアクセスするアクイジション番号を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] count	アクセスするアクイジション番号

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ScGetAcqData、ScAcqDataLength、ScGetTriggerTime でアクセスするアクイジション番号を取得します。

注意 :

SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

ScSetAcqCount

概要 :

トリガモードでアクセスするアクイジション番号を設定する

書式 :

[C++]
ScResult ScSetAcqCount(ScHandle hndl, __int64 count);
[C#]
int ScSetAcqCount(int hndl, long count);

引数 :

[IN] hndl	測定器ハンドル
[IN] count	アクセスするアクイジション番号

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ScGetAcqData、ScAcqDataLength、ScGetTriggerTime でアクセスするアクイジション番号を設定します。

注意 :

SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

ScGetTriggerTime

概要 :

トリガ時刻を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] buff	トリガ時刻文字列

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

ScSetAcqCount で設定したアクイジション番号のトリガ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 ～ 31）, 時（0 ～ 23）, 分（0 ～ 59）, 秒（0 ～ 59）, ナノ秒（0 ～ 999999999）

注意 :

SC_TRIGGER および SC_TRIGGER_ASYNC モード指定時に使用できます。

ScResumeAcquisition

概要 :

トリガ同期モードで波形取り込みを再開する

書式 :

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

トリガでの波形取り込み中に同期待ちで波形取り込みを停止している DL950/SL2000 の波形取り込みを再開します。

注意 :

DL950/SL2000 が同期待ちで停止していない場合には何も起こりません。
DL950/SL2000 がタイムアウトを検出した場合、本コマンドを受信しなくても波形取り込みを再開します。
SC_TRIGGER モード指定時に使用できます。

ScSetTriggerTimeout

概要 :

トリガ同期モードで DL950/SL2000 の同期タイムアウト値を設定する

書式 :

[C++]
ScResult ScSetTriggerTimeout(ScHandle hndl, int timeout);
[C#]
int ScSetTriggerTimeout(int hndl, int timeout);

引数 :

[IN] hndl	測定器ハンドル
[IN] timeout	タイムアウト値（0 ～ 497664 秒、初期値 : 600 秒）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

トリガ同期モードで DL950/SL2000 との同期処理おいて、PC からの波形取り込み再開コマンドに対するタイムアウト値（単位 : 秒）を設定します。
0 を設定した場合、PC からの波形取り込み再開コマンドが送られるまで待ち続けます。
1 ～ 497664 を設定した場合、設定されている時間が経過すると、PC からの波形取り込み再開コマンドを待たずに次の波形取り込みを実行します。
なお、PC からの波形取り込み再開コマンドを送る前に以下の処理を実行すると DL950/SL2000 でのタイマーはリスタートされます。
ScGetAcqData、ScGetAcqDataLength、ScGetLatchAcqCount、ScGetAcqCount、ScSetAcqCount、ScGetTriggerTime

注意 :

同期待ちで停止していない場合には何も起こりません。
SC_TRIGGER モード指定時に使用できます。

ScGetTriggerTimeout

概要 :

トリガ同期モードで DL950/SL2000 の同期タイムアウト値を取得する

書式 :

[C++]
ScResult ScGetTriggerTimeout(ScHandle hndl, int *timeout);
[C#]
int ScSetTriggerTimeout(int hndl, out int timeout);

引数 :

[IN] hndl	測定器ハンドル
[OUT] timeout	タイムアウト値（0 ～ 497664）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

トリガ同期モードで DL950/SL2000 との同期処理おいて、PC からの波形取り込みコマンドに対するタイムアウト値（単位 : 秒）を取得します。

ScGetMaxHistoryCount

概要 :

トリガモードでの最大ヒストリ数を取得する

書式 :

[C++]
ScResult ScGetMaxHistoryCount(ScHandle hndl, int *count);
[C#]
int ScGetMaxHistoryCount(int hndl, out int count);

引数 :

[IN] hndl	測定器ハンドル
[OUT] count	最大ヒストリ数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

トリガモードで DL950/SL2000 の本体内に保存できる最大ヒストリ数を取得します。

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 :

波形の取り込み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。

ScGetChannelSamplingRate

概要 :

チャネルのサンプル周波数を取得する

書式 :

[C++]
ScResult ScGetChannelSamplingRate(ScHandle hndl, int chNo, int subChNo,
double* srate);
[C#]
int ScGetChannelSamplingRate(int hndlhNo, int chNo, int subChNo, out double srate);

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] srate	サンプル周波数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

チャネルのサンプル周波数を取得します。

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
[IN] subChNo	サブチャネル番号（無い場合は 0 を指定）
[OUT] bits	データビット長（1 ～ 32）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

取得するチャネルデータ（AD 値の有効な）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
[IN] subChNo	サブチャネル番号（無い場合は 0 を指定）
[OUT] gain	ゲイン値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

取得する波形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
[IN] subChNo	サブチャネル番号（無い場合は 0 を指定）
[OUT] offset	オフセット値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

取得する波形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。

ScGetChannelScale

概要 :

チャネルの表示スケールの上限値 / 下限値を取得する

書式 :

[C++]
ScResult ScGetChannelScale(ScHandle hndl, int chNo, int subChNo, double* upper,
double* lower);
[C#]
int ScGetChannelScale(int hndl, int chNo, int subChNo, out double upper,
out double lower);

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] upper	表示スケールの上限値
[OUT] lower	表示スケールの下限値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

DL950/SL2000 の画面上に設定されている表示スケールの上限値 / 下限値を取得します。

ScGetChannelType

概要 :

チャネルのデータ種別を取得する

書式 :

[C++]
ScResult ScGetChannelType(ScHandle hndl, int chNo, int subChNo, char* type);
[C#]
int ScGetChannelType(int hndl, int chNo, int subChNo, out string type);

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] type	データ種別

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

取得する測定データのデータ種別を文字列として取得します。
※文字列は通信コマンドの応答仕様に準じるため通常は省略形となります。

ANALog	波形データがアナログ形式（実値＝データ * Gain + Offset）
LOGic	波形データが Logic 形式のデータ
FLOat	波形データが単精度浮動小数点数形式（RMath チャネルが該当します）
TIMe	32 ビットの UNIX 時刻と 32 ビットのナノ秒単位の秒未満の時刻

（G5 のサブチャネル番号が 63、または GPS のサブチャネル番号 7 のデータが該当）

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。
登録可能なイベントはフリーランモードの「オーバーランイベント」及びトリガ同期モードの「トリガエンドイベント」になります。
handleEventScCallListener をオーバーライドすることによって、オーバーラン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。
handleEventScTrigEnd をオーバーライドすることによって、トリガエンド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 :

オーバーランイベントは 10GEther 以外で接続されている場合に有効です。
.NET 版（C#）では使えません。
使用例（フリーランの場合）:

```
class cMyEvent : public ScEventListener {
public:
  virtual void handleEventScCallListener(ScHandle hndl,
   __int64 reserve);
};
cMyEvent* ep = new cMyEvent();
ScAddEventListener(hndl, ep);
```

使用例（トリガ同期モードの場合）:

```
class cMyEvent : public ScEventListener {
public:
  virtual void handleEventScTrigEnd(ScHandle hndl);
};
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, int type);

引数 :

[IN] hndl	測定器ハンドル
[IN] func	コールバックメソッド
[IN] type	イベント種別

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

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。
登録可能なイベントはフリーランモードの「オーバーランイベント」およびトリガ同期モードの「トリガエンドイベント」となります。
イベント種別は、「SC_EVENTTYPE_OVERRUN」、「SC_EVENTTYPE_TRIGGEREND」になります。

注意 :

オーバーランイベントは 10GEther 以外で接続されている場合に有効です。
C++ では使えません。

使用例 :

```
private void overrunCallback(int hndl, int type)
{
  ....
}
if (api.ScAddCallback(hndl, overrunCallback,
 SC_EVENTTYPE_OVERRUN) != ScSDK.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 :

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。

4.3 フラッシュアクイジションデータアクセスライブラリ機能用 API

ScGetFAcqCount

概要 :

測定器に保存されているフラッシュアクイジションの波形データのファイル数を取得する

書式 :

[C++]
ScResult ScGetFAcqCount(ScHandle hndl, int* count);
[C#]
int ScGetFAcqCount(int hndl, out int count);

引数 :

[IN] hndl	測定器ハンドル
[OUT] count	保存されているフラッシュアクイジションの波形データのファイル数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

測定器に保存されているフラッシュアクイジションの波形データのファイル数を取得します。

ScGetFAcqFileName

概要 :

測定器に保存されているフラッシュアクイジションの波形データのファイル名を取得する

書式 :

[C++]
ScResult ScGetFAcqFileName(ScHandle hndl, int count, char* name);
[C#]
int ScGetFAcqFileName(int hndl, int count, out string name);

引数 :

[IN] hndl	測定器ハンドル
[IN] count	フラッシュアクイジション番号
[OUT] name	ファイル名

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_NODATA	該当ファイルなし

詳細 :

測定器に保存されているフラッシュアクイジションの波形データのファイル名を取得します。countにはフラッシュアクイジション番号（ScGetFacqCountで保存されているファイル数を確認できます）を指定します。保存されている一番古いフラッシュアクイジションの波形データが 1 となります。

注意 :

フラッシュアクイジション番号は 1 ～ ScGetFacqCount で取得できる値までの範囲となります。

ScOpenFAcqData

概要 :

オープンするフラッシュアクイジションの波形データのファイル名を指定する

書式 :

[C++]
ScResult ScOpenFAcqData(ScHandle hndl, char* name , int* result);
[C#]
int ScOpenFAcqData(int hndl, string name, out int result);

引数 :

[IN] hndl	測定器ハンドル
[IN] name	ファイル名
[OUT] result	オープンの可否

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_USE_ACQMEMORY	アクイジションメモリーに波形データが存在している
SC_ERR_RUNNING	測定中エラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

PC にデータの転送を行うフラッシュアクイジションの波形データのファイルをオープンします。
ファイルがオープンできた場合は result に 0 が保存されます。失敗した場合は 0 以外が保存されます。
ファイル名には、ScGetFAcqFileName で取得したものを設定します。

注意 :

オープンは同時に複数行うことはできません。複数のデータファイルを取得したい場合はそれぞれのアクイジションの波形データに対してオープン / クローズおよびデータの転送処理を行うようにしてください。
フラッシュアクイジションの波形データを転送する場合、アクイジションメモリーを一時バッファーとして使用します。そのためアクイジションメモリーに波形データが存在している場合、アクイジションメモリーをクリアしないとオープンできません。アクイジションメモリーのクリアには ScClearAcqMemory を実行してください。

ScCloseFAcqData

概要 :

オープン中の波形データのファイルをクローズする

書式 :

[C++]
ScResult ScCloseFAcqData(ScHandle hndl);
[C#]
int ScCloseFAcqData(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
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScOpenFAcqData でオープンした波形データをクローズします。

ScClearAcqMemory

概要 :

アクイジションメモリーにある波形データを消去する

書式 :

[C++]
ScResult ScClearAcqMemory(ScHandle hndl);
[C#]
int ScClearAcqMemory(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
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

アクイジションメモリーにある波形データを消去します。

注意 :

メディアに保存されていない波形データは消去されますので、本 API を実行する前に必要に応じて保存を行ってください。

ScGetFAcqStartTime

概要 :

オープン中の波形データの測定開始時刻を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] startTime	測定開始時刻

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScOpenFAcqData でオープンした波形データの測定開始時刻を文字列で取得します。
文字列の形式は次のようになります。秒未満はナノ秒単位となります。
出力形式 : "2020/10/01 12:34:56:123456789"

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqTimeBase

概要 :

オープン中の波形データのタイムベース設定を取得する

書式 :

[C++]
ScResult ScGetFAcqTimeBase(ScHandle hndl, int* timeBase);
[C#]
int ScGetFAcqTimeBase(int hndl, out int timeBase);

引数 :

[IN] hndl	測定器ハンドル
[OUT] timeBase	タイムベース設定（0: 内部、1: 外部）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScOpenFAcqData でオープンした波形データのタイムベース設定を取得します。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqComment

概要 :

オープン中の波形データのコメントを取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[OUT] comment	コメント文字列

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScOpenFAcqData でオープンした波形データのコメント文字列を取得します。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelCount

概要 :

オープン中の波形データに保存されている総チャネル数を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelCount(ScHandle hndl, int* count);
[C#]
int ScGetFAcqChannelCount(int hndl, out int count);

引数 :

[IN] hndl	測定器ハンドル
[OUT] count	総チャネル数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

オープンした波形データに保存されている総チャネル数を取得します。ここで取得した総チャネル数をもとに ScGetFAcqChannleNumber で保存されているチャネル番号の取得をします。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelNumber

概要 :

オープンした波形データに保存されているチャネル番号を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] index	インデックス番号
[OUT] chNo	チャネル番号
[OUT] subChNo	サブチャネル番号

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

波形データに保存されているチャネル番号を取得します。指定したインデックス番号をもとにチャネル番号を取得します。インデックス番号は ScGetFAcqChannelCount で取得した総チャネル数をもとに設定します。
チャネル番号は 1 ～ 32 の範囲となります。
RMath チャネルのチャネル番号は 17 ～ 32 範囲となります。
GPS チャネルのチャネル番号は 17 となります。
ここで取得したチャネル番号とサブチャネル番号は、データ取得やチャネル情報の取得用 API で使用します。

注意 :

インデックス番号は 1 ～ ScGetFacqChannelCount で取得できる値までの範囲となります。
サブチャネル番号が存在しないモジュールでもサブチャネル番号は 1 となります。
ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelBits

概要 :

指定チャネルの 1 データあたりのビット数を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelBits(ScHandle hndl, int chNo, int subChNo, int* bits);
[C#]
int ScGetFAcqChannelBits(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 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] bits	データビット数（1 ～ 32）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルの 1 データ（AD 値）あたりのビット数を取得できます。

注意 :

ここでのビット数はモジュールの分解能とは異なります。例えば電圧モジュールなどのアナログモジュールのチャネルであれば 16、RMath チャネルの場合は 32 となります。
Logic モジュール（720230）でも 16 となります。
ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelGain

概要 :

指定チャネルのゲイン値を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelGain(ScHandle hndl, int chNo, int subChNo, double* gain);
[C#]
int ScGetFAcqChannelGain(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 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] gain	サンプルインターバル値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのゲイン値を取得できます。ゲイン値およびオフセット値から、取得したチャネルのデータの物理値を以下の計算式で求めます。
物理値＝データ× Gain+Offset

注意 :

ゲイン値およびオフセット値には、リニアスケールの設定が反映されています。
物理値への換算はデータ種別（ScGetFAcqChannelType）が "ANALog" の場合に必要になります。
ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelHOffset

概要 :

指定チャネルの測定開始時刻からのオフセット時間（単位 : 秒）を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelHoffset(ScHandle hndl, int chNo, int subChNo, double* hOffset);
[C#]
int ScGetFAcqChannelHOffset(int hndl, int chNo, int subChNo, out double hOffset);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] hOffset	測定開始時刻からのオフセット値（単位 : 秒）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルの波形取り込みを開始した最初のデータと測定開始時刻からのオフセット時間（位相差）を取得できます。タイベース設定により以下の意味になります。
内部サンプルの場合 : 測定開始時刻からのオフセット時間（単位 : 秒）
外部サンプルの場合 : 測定開始点からの位相差点数

内部サンプルの場合、DL950/SL2000 本体の画面右上に表示される基準のサンプルレート（レコード長と T/Div の設定で決まります）と、チャネルのサンプルレートが異なる場合に、基準となるサンプルとチャネルのサンプルに位相差が発生する場合があります。その場合に、チャネルのデータの取り込みを開始し始めたデータと基準のサンプルとなる測定開始時刻との位相差をオフセット時間として取得します。チャネルのサンプレートが基準と同じ場合は、0 となります。

外部サンプルの場合は、サブチャネルが存在するモジュールでは基準となる外部サンプルをサブチャネルの数に応じたサンプル間隔でデータを取り込みます。その場合に、チャネルの取り込みを開始したデータと基準となる外部サンプルで取り込みを開始したデータからの位相差点数を示します。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelHResolution

概要 :

指定チャネルのサンプルインターバルの値を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] hResolution	サンプルインターバル値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのサンプルインターバルの値を取得できます。タイベース設定により以下の意味になります。
内部サンプルの場合 : チャネルのサンプルインターバル（1 / サンプレート）（単位 : 秒）
外部サンプルの場合 : 1 / PulseRotate

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelLabel

概要 :

指定チャネルのチャネルラベル名を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelLabel(ScHandle hndl, int chNo, int subChNo, char* label);
[C#]
int ScGetFAcqChannelLabel(int hndl, int chNo, int subChNo, out string label);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] label	チャネルラベル

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのラベル名を取得できます。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelLogicBits

概要 :

指定チャネルの Logic の有効ビット数を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelLogicBits(ScHandle hndl, int chNo, int subChNo, int* bits);
[C#]
int ScGetFAcqChannelLogicBits(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 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] bits	Logic の有効ビット数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルの Logic の有効ビット数を取得できます。モジュールが 720230 の場合は、常に 8 となります。CAN モジュールなどでは、設定したビット数となります。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelLogicLabel

概要 :

指定チャネルの Logic のビットラベル名を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelLogicLabel(ScHandle hndl, int chNo, int subChNo, int bitNo,
char* label);
[C#]
int ScGetFAcqChannelLogicLabel(int hndl, int chNo, int subChNo, int bitNo,
out string label);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[IN] bitNo	ビット番号（1 ～ 8）
[OUT] label	ビットラベル

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルの Logic のビットラベル名を取得できます。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelOffset

概要 :

指定チャネルのオフセット値を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelOffset(ScHandle hndl, int chNo, int subChNo, double* offset);
[C#]
int ScGetFAcqChannelOffset(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 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] offset	オフセット値

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのオフセット値を取得できます。ゲイン値およびオフセット値から、
取得したデータの物理値を以下の計算式で求めます。
物理値＝データ * Gain + Offset

注意 :

ゲイン値およびオフセット値には、リニアスケールの設定が反映されています。
物理値への換算はデータ種別（ScGetFAcqChannelType）が "ANALog" の場合に必要になります。
ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelSign

概要 :

指定チャネルのデータの符号を取得する

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] sign	符号（0: 符号なし、1: 符号あり）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのデータ符号を取得できます。通常の電圧モジュールなどのアナログモジュールは符号ありとなります。CAN モジュール系など符号が設定できる場合は、設定に応じた値となります。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelType

概要 :

指定チャネルのデータ種別を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelType(ScHandle hndl, int chNo, int subChNo, char* type);
[C#]
int ScGetFAcqChannelType(int hndl, int chNo, int subChNo, out string type);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] type	データ種別

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルのデータ種別を文字列として取得します。
※文字列は通信コマンドの応答仕様に準ずるため通常は省略系となります。
ANALog　波形データがアナログ形式（実値＝データ * Gain + Offset）
LOGic　波形データが Logic 形式のデータ
FLOat　波形データが単精度浮動小数点数形式（RMath チャネルが該当します）
TIMe　32 ビットの UNIX 時刻と 32 ビットのナノ秒単位の秒以下の時刻
（G5 のサブチャネル番号が 63、または GPS のサブチャネル番号 7 のデータが該当）

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqChannelUnit

概要 :

指定チャネルの単位文字列名を取得する

書式 :

[C++]
ScResult ScGetFAcqChannelUnit(ScHandle hndl, int chNo, int subChNo, char* unit);
[C#]
int ScGetFAcqChannelUnit(int hndl, int chNo, int subChNo, out string unit);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）
[OUT] unit	単位文字列

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

指定したチャネルの単位文字列を取得できます。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScSetFAcqChannelNumber

概要 :

取得する波形データのチャネル番号を設定する

書式 :

[C++]
ScResult ScSetFAcqChannelNumber(ScHandle hndl, int chNo, int subChNo);
[C#]
int ScSetFAcqChannelNumber(int hndl, int chNo, int subChNo);

引数 :

[IN] hndl	測定器ハンドル
[IN] chNo	チャネル番号（1 ～ 32）
[IN] subChNo	サブチャネル番号（1 ～ 64、無い場合は 0 を指定）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー
SC_ERR_PARAMETER	パラメータエラー

詳細 :

取得する波形データのチャネル番号を設定します。この API は、ScGetFAcqDataLength、ScGetFAcqData の API 対して有効となります。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqDataLength

概要 :

設定されているチャネルのデータ点数を取得する

書式 :

[C++]
ScResult ScGetFAcqDataLength(ScHandle hndl, __int64* length);
[C#]
int ScGetFAcqDataLength(int hndl, out long length);

引数 :

[IN] hndl	測定器ハンドル
[OUT] length	データ点数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScSetFacqChannelNumber で設定したチャネルのデータ点数を取得できます。

注意 :

ScOpenFAcqData でオープンした波形データに対して有効です。

ScGetFAcqData

概要 :

設定されているチャネルのデータを取得する

書式 :

[C++]
ScResult ScGetFAcqData(ScHandle hndl, char* buff, int buffLen, int* dataLen);
[C#]
int ScGetFAcqData(int hndl, ref DT[] buff, int buffLen, out int dataLen);

引数 :

[IN] hndl	測定器ハンドル
[OUT] buff	データを保存するバッファー
[IN] buffLen	データを保存するバッファーの長さ（単位 : バイト）
[OUT] dataLen	保存したデータの長さ（単位 : バイト）

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScSetFacqChannelNumber で設定したチャネルのデータを取得できます。1 回の転送で必要なバッファーサイズは ScSetFAcqDataSize の設定によります。バッファーサイズよりも転送するチャネルの波形データが大きい場合は、本コマンドを複数回連続して実行することで連続した波形データを取得することができます。
プログラミング時の Tips:
データ点数およびビット数から本コマンドを実行するループ回数を計算する方法もありますが、dataLen が 0 になるまでループさせる方法もあります。

注意 :

ScGetFAcqDataLength で取得する値はデータ点数となり、本コマンドで使用しているのはバイト換算したバッファーサイズとなります。データ点数からサイズ換算する場合は、ScGetFacqChannelBits で取得できる 1 点のあたりのビット数をもとに換算してください。
該当チャネルの波形データをすべて取得したあとに、本コマンドを実行すると dataLen の値が 0 となります。
該当チャネルの波形データの取得途中に、ScSetFacqChannelNumber を実行すると状態がリセットされ変更後のチャネルのデータを最初から転送することになります。
ScOpenFAcqData でオープンした波形データに対して有効です。

ScSetFAcqDataSize

概要 :

一回の送信可能な最大バッファーサイズを設定する

書式 :

[C++]
ScResult ScSetFAcqDataSize(ScHandle hndl, int size);
[C#]
int ScSetFAcqDataSize(int hndl, int size);

引数 :

[IN] hndl	測定器ハンドル
[IN] size	サイズ SC_SIZE_16MB / SC_SIZE_32MB / SC_SIZE_64MB / SC_SIZE_128MB / SC_SIZE_256MB / SC_SIZE_512MB

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_UNOPENED	ファイル未指定
SC_ERR_PARAMETER	サイズ指定エラー
SC_ERR_NOTAPPLICABLE	対象機器エラー

詳細 :

ScGetFAcqData を 1 回実行する際に DL950/SL2000 から送信可能なバッファーサイズを設定します。
16、32、64、128、256、512MiB から設定可能です。デフォルトは 16MiB です。

注意 :

このサイズは、バッファーサイズとなり、データ点数とは異なります。
ScOpenFAcqData でオープンした波形データに対して有効です。

4.4　ファイル操作・転送機能用 API

ScSetCurrentDrive

概要 :

カレントドライブを設定する

書式 :

[C++]
ScResult ScSetCurrentDrive(ScHandle hndl, int drive);
[C#]
int ScSetCurrentDrive(int hndl, int drive);

引数 :

[IN] hndl	測定器ハンドル	
[IN] drive	カレントドライブ	
	SC_DRIVE_IDRIVE	IDRive
	SC_DRIVE_NETWORK	NETWork
	SC_DRIVE_SD	SD
	SC_DRIVE_USB_0	USB-0
	SC_DRIVE_USB_1	USB-1
	SC_DRIVE_FLASH	FLASh

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	カレントドライブの指定エラー

詳細 :

drive で指定したドライブをカレントドライブに設定します。

ScGetCurrentDrive

概要 :

設定されているカレントドライブの問い合わせ

書式 :

[C++]
ScResult ScGetCurrentDrive(ScHandle hndl, int* drive);
[C#]
int ScGetCurrentDrive(int hndl, out int drive);

引数 :

[IN] hndl	測定器ハンドル	
[OUT] drive	カレントドライブ	
	SC_DRIVE_IDRIVE	IDRive
	SC_DRIVE_NETWORK	NETWork
	SC_DRIVE_SD	SD
	SC_DRIVE_USB_0	USB-0
	SC_DRIVE_USB_1	USB-1
	SC_DRIVE_FLASH	FLASh

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

設定されているカレントドライブの問い合わせを行います。

ScSetCurrentDirectory

概要 :

カレントディレクトリの設定

書式 :

[C++]
ScResult ScSetCurrentDirectory(ScHandle hndl, char* pathName);
[C#]
int ScSetCurrentDirectory(int hndl, DT[] buff, string pathName);

引数 :

[IN] hndl	測定器ハンドル	
[OUT] pathName	パス名	
	（例）	
	"/abcdefg/efghi"	→ 絶対パス指定
	"jklmn"	→ 相対パス指定
	".."	→ 親ディレクトリへ
	"/"	→ ルートディレクトリへ

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	パス名空文字エラー

詳細 :

カレントディレクトリを変更します。

ScGetCurrentDirectory

概要 :

カレントディレクトリの取得

書式 :

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

引数 :

[IN] hndl	測定器ハンドル	
[OUT] pathName	パス名	
	（例）	
	"/abcdefg/efghi"	

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

カレントディレクトリを取得します。

ScGetFileNum

概要 :

ファイル数の取得

書式 :

[C++]
ScResult ScGetFileNum(ScHandle hndl, int* number);
[C#]
int ScGetFileNum(int hndl, out int number);

引数 :

[IN] hndl	測定器ハンドル
[OUT] number	ファイル数

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

カレントディレクトリに保存されているファイル数を取得します。

注意 :

ScGetFileInfo でファイル情報を取得する場合は、目的のカレントディレクトリでこの API を事前に実行してください。

ScGetFileInfo

概要 :

ファイル情報の取得

書式 :

[C++]
ScResult ScGetFileInfo(ScHandle hndl, int index, FileInfo* info);
[C#]
int ScGetFileInfo(int hndl, int index, out FileInfo info);

引数 :

[IN] hndl	測定器ハンドル
[IN] index	ファイル番号
[OUT] info	ファイル情報

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー

詳細 :

カレントディレクトリにある index 番号のファイルの情報を取得します。

注意 :

ファイル情報を取得する前に ScGetFileNum でファイル数を取得してください。

ScDeleteFile

概要 :

ファイルの削除

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] name	ファイル名

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名空文字エラー

詳細 :

カレントディレクトリにある name で指定したファイルを削除します。

ScDownloadFile

概要 :

ファイルの取得

書式 :

[C++]
ScResult ScDownloadFile(ScHandle hndl, char* srcFileName, char* dstFileName);
[C#]
int ScDownloadFile(int hndl, string srcFileName, string dstFileName);

引数 :

[IN] hndl	測定器ハンドル
[IN] srcFileName	コピー元のファイル名 （例） "stuvw.xyz"
[IN] dstFileName	コピー先のファイル名（PC 側） （例） "C:\ScSDK\stuvw.xyz"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名空文字エラー

詳細 :

カレントディレクトリにある、srcFileName で指定したファイルを取得します。

ScUploadFile

概要 :

ファイルの送信

書式 :

[C++]
ScResult ScUploadFile(ScHandle hndl, char* srcFileName, char* dstFileName);
[C#]
int ScUploadFile(int hndl, string srcFileName, string dstFileName);

引数 :

[IN] hndl	測定器ハンドル
[IN] srcFileName	コピー元のファイル名（PC 側） （例） "C:\ScSDK\stuvw.xyz"
[IN] dstFileName	コピー先のファイル名（ユニット側） （例） "stuvw.xyz"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名空文字エラー

詳細 :

カレントディレクトリに srcFileName で指定したファイルを保存します。

注意 :

ファイルサイズが 0 の場合はエラーになります。

ScSaveTriggerWDF

概要 :

トリガ波形のファイルの保存

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] dstFileName	コピー先のファイル名（PC 側） （例） "C:\ScSDK\stuvw.WDF"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_MODE	モード違いエラー
SC_ERR_PARAMETER	ファイル名空文字エラー
SC_ERR_RUNNING	測定中エラー

詳細 :

表示されているトリガの波形データを WDF ファイルとして PC 側に保存します。

注意 :

DL950/SL2000 に波形が表示されていない場合はエラーを返します。
リアルタイム記録のデータを保存する場合は、ScTmcSetTimeout を使用して TMCTL のタイムアウト時間を調整してください。
ファイル保存の設定は、本体の設定に依存します。

ScSaveFreeRunWDF

概要 :

フリーラン波形のファイルの保存

書式 :

[C++]
ScResult ScSaveFreeRunWDF(ScHandle hndl, INT64 startPos, INT64 count, char *srcFileName);
[C#]
int ScSaveFreeRunWDF(int hndl, Int64 startPos, Int64 count, string srcFileName);

引数 :

[IN] hndl	測定器ハンドル
[IN] startPos	保存開始位置
[IN] count	保存点数（最大値は ScGetFreeRunDataLength で求めた値）
[IN] srcFileName	コピー先のファイル名（PC 側） （例） "C:\ScSDK\stuvw.WDF"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_MODE	モード違いエラー
SC_ERR_PARAMETER	ファイル名空文字エラー
SC_ERR_RUNNING	測定中エラー

詳細 :

表示されているフリーランの波形データを WDF ファイルとして PC 側に保存します。

注意 :

保存可能な有効点数は、ScGetFreeRunDataLength で取得することができます。

ScSaveSetup

概要 :

設定ファイルの取得

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] fileName	保存する設定ファイル名（PC 側） （例） "C:\ScSDK\stuvw.SET"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名空文字エラー

詳細 :

DL950/SL2000 の設定を PC 側に設定ファイルとして保存します。

ScLoadSetup

概要 :

設定ファイルの設定

書式 :

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

引数 :

[IN] hndl	測定器ハンドル
[IN] fileName	設定ファイル名（PC 側） （例） "C:\ScSDK\stuvw.SET"

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名空文字エラー

詳細 :

fileName で指定した設定ファイルの設定を DL950/SL2000 に反映します。

注意 :

ファイルサイズが 0 の場合はエラーになります。

ScGetFileList

概要 :

ファイル名一覧の取得

書式 :

[C++]
ScResult ScGetFileList(ScHandle hndl , char** fileList, int max, int extension, int sort,
int* fileCount);
[C#]
int ScGetFileList(int hndl, out string[] fileList, int max, int extension, int sort,
out int fileCount);

引数 :

[IN] hndl	測定器ハンドル	
[OUT] fileList	ファイル名一覧	
[IN] max	ファイル名一覧の最大サイズ	
[IN] extension	ファイル拡張子	
	SC_FILE_ETE_ALL	*.*
	SC_FILE_ETE_SET	*.SET
	SC_FILE_ETE_WDF	*.WDF
	SC_FILE_ETE_BMP	*.BMP
	SC_FILE_ETE_PNG	*.PNG
	SC_FILE_ETE_JPG	*.JPG
	SC_FILE_ETE_SNP	*.SNP
	SC_FILE_ETE_SBL	*.SBL
	SC_FILE_ETE_CSV	*.CSV
	SC_FILE_ETE_MAT	*.MAT
[IN] sort	ソート順	
	SC_SORT_NAME_ASC	名前昇順
	SC_SORT_NAME_DESC	名前降順
	SC_SORT_DATE_ASC	更新日時昇順
	SC_SORT_DATE_DESC	更新日時降順
	SC_SORT_SIZE_ASC	ファイルサイズ昇順
	SC_SORT_SIZE_DESC	ファイルサイズ降順
[OUT] fileCount	リストに保存されたファイル数	

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名一覧の最大サイズエラー、 ファイル拡張子指定エラー

詳細 :

カレントディレクトリ内の指定したファイル拡張子のファイル名一覧を取得します。
max は配列の最大サイズを指定します。

ScGetFileInfoList

概要 :

ファイル情報一覧の取得

書式 :

[C++]
ScResult ScGetFileInfoList(ScHandle hndl, FileInfo* infoList, int max, int extension,
int sort, int* fileCount);
[C#]
int ScGetFileInfoList(int hndl, out FileInfo[] infoList, int max, int extension,
int sort, out int fileCount);

引数 :

[IN] hndl	測定器ハンドル	
[OUT] fileList	ファイル情報一覧	
[IN] max	ファイル情報一覧の最大サイズ	
[IN] extension	ファイル拡張子	
	SC_FILE_ETE_ALL	*.*
	SC_FILE_ETE_SET	*.SET
	SC_FILE_ETE_WDF	*.WDF
	SC_FILE_ETE_BMP	*.BMP
	SC_FILE_ETE_PNG	*.PNG
	SC_FILE_ETE_JPG	*.JPG
	SC_FILE_ETE_SNP	*.SNP
	SC_FILE_ETE_SBL	*.SBL
	SC_FILE_ETE_CSV	*.CSV
	SC_FILE_ETE_MAT	*.MAT
[IN] sort	ソート順	
	SC_SORT_NAME_ASC	名前昇順
	SC_SORT_NAME_DESC	名前降順
	SC_SORT_DATE_ASC	更新日時昇順
	SC_SORT_DATE_DESC	更新日時降順
	SC_SORT_SIZE_ASC	ファイルサイズ昇順
	SC_SORT_SIZE_DESC	ファイルサイズ降順
[OUT] fileCount	一覧に保存されたファイル数	

戻り値 :

SC_SUCCESS	成功
SC_ERROR	エラー
SC_ERR_PARAMETER	ファイル名一覧の最大サイズエラー、 ファイル拡張子指定エラー

詳細 :

カレントディレクトリ内の指定したファイル拡張子のファイル情報一覧を取得します。
max は配列の最大サイズを指定します。

4.5 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）と同じフォルダーに以下の DLL を配置してください。

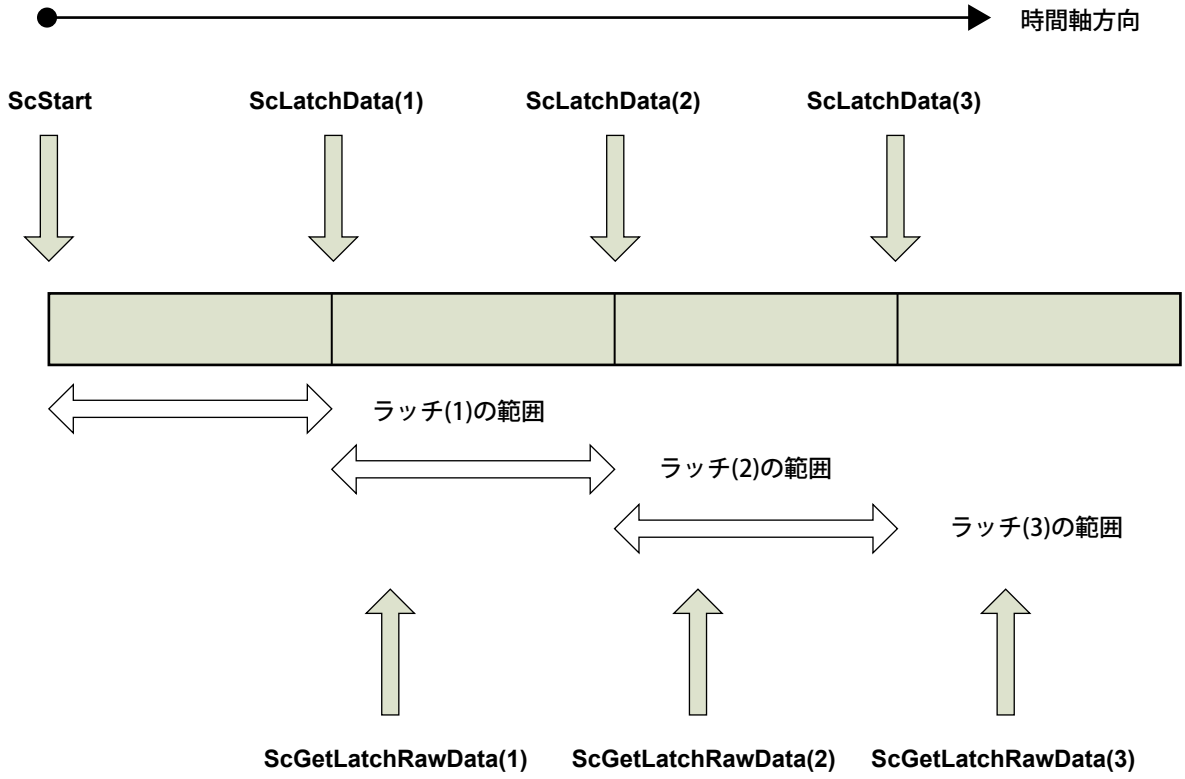
プロジェクト	C++（アンマネージアプリ）		C#（マネージアプリ）		
アーキテクチャ	32bit	64bit	32bit	64bit	Any CPU
ScSDK.dll	○		○		○
ScSDK 64.dll		○		○	○
ScSDK Net.dll			○	○	○
tmctl.dll	○		○		○
tmctl64.dll		○		○	○

5.1　データアクイジション機能

フリーランモードについて

本 API と DL950/SL2000 を使用したフリーランモードは以下のような仕組みで動作します。

DL950/SL2000 は波形の取り込み開始（ScStart）を受け付けてから波形の取り込み停止（ScStop）を受け付けるまで波形を取得します。波形データは、本体のアクイジションメモリーに一時的に保存されます。
波形の取り込み中に、API からラッチ（ScLatchData）および波形取得（ScGetLatchRawData）を繰り返し実行します。取得できる波形データはラッチからラッチまでの間のデータになります。
また、1 回のラッチで取得できる波形データは、すべてのチャネルのデータが含まれた状態で DL950/SL2000 本体から API まで送られます。そのため、API の使用するバッファーサイズには注意が必要となります。



サンプリングレートと回線種別、接続モードについて

本 API と DL950/SL2000 の接続において回線種別として複数の中から選択できますが、使用する回線により波形の取り込みが可能なサンプリングレートが異なります。

10G による高速転送を行う場合

接続時に回線種別として HiSLIP（SC_WIRE_HISLIP）を選択します。この場合、DL950/SL2000 では最大 10MS/s × 16ch での波形の取り込みが可能です。

上記以外の場合

10G の HiSLIP 接続以外を選択している場合は、DL950/SL2000 では最大 200kS/s × 16ch での波形の取り込みが可能です。

波形の取り込みをしているサンプリングレートが早い場合、データ取得の間隔が長くなると波形データが DL950/SL2000 本体内のメモリーに上書きされる場合があります。

必要なメモリーサイズについて

フリーランモードでデータを取得する際は、「ScGetLatchRawData のデータ構造について」に記載してある形式で波形の取り込みをしているすべてのチャネルが一括で送られてきます。ScGetLatchRawData で設定するバッファーサイズは、以下のパラメータから計算を行い、必要なメモリーサイズを設定する必要があります。

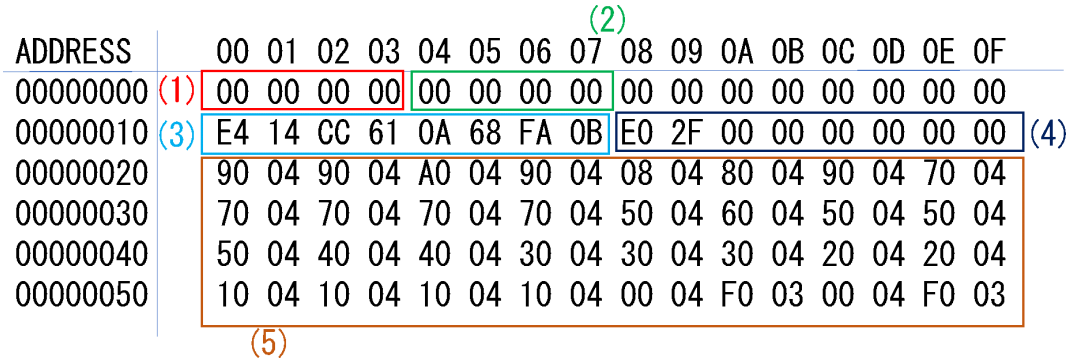
- 使用しているチャネル数
- サンプリングレート
- ラッチ間隔

たとえば、200kS/s で 16ch（電圧モジュール）でフリーランモードで実行した場合、波形データのみで毎秒 6400000byte（=400000byte × 16ch）必要になります。
さらに波形の取り込みをしているチャネルごとにヘッダー情報として 32byte のデータが必要なります。
よって、合計毎秒 6400512byte（=6400000byte+32byte × 16ch）になります。

ScGetLatchRawData のデータ構造について

フリーランモードの場合、DL950/SL2000 から送られてくるデータは、波形の取り込みをしている全チャネルのデータが一括して送られてきます。
送られてくるデータ形式は以下のようになります。各チャネルが以下の形式で連続した状態となります。データはすべて LittleEndian 形式となります。

1	チャネル番号（4Byte）	0 ～ 31*1	（1）の枠線部分
2	サブチャネル番号（4Byte）	0 ～ 63*1、*2	（2）の枠線部分
3	予約（8Byte）		
4	先頭データの時刻（8Byte）	Unix Time（4Byte）＋ Tick（4Byte、nsec 単位（0 ～ 999999999））*4	（3）の枠線部分
5	データサイズ（8Byte）	0 ～ データサイズは転送する ACQ データ点数をバイト数に換算し直したものになります。*3	（4）の枠線部分
6	ACQ データ	ACQ の 1 点のデータサイズは「ScGetChannelBits」で確認できます。*3	（5）の枠線部分



*1 チャネル番号、サブチャネル番号はともに 0 オリジンとなります。（波形の取り込みチャネルが CH1 の場合は’ 0’ となります、RMath1 の場合は’ 16’ となります。）

*2 モジュールが 720240、720241、720242、720243 の場合、サブチャネル番号ではなく有効なサブチャネル数で決まる値となります。
たとえばサブチャネル 1 とサブチャネル 3 が測定対象でサブチャネル 2 が無効の場合、’ 0’ がサブチャネル 1 と’ 1’ がサブチャネル 3 となります。

*3 基本的に通常のモジュールはデータの 1 点は 2 バイトとなります。CAN などで 17 ビット以上を設定している場合、4 バイトとなります。RMath チャネルはデータが浮動小数点形式のため 4 バイトとなります。電力演算、高調波演算の各サブチャネルはデータが浮動小数点形式のため 4 バイトとなります。GPS の各サブチャネルは 32 ビッ整数型となるため 4 バイトとなります。
電力演算、高調波演算、GPS の時刻情報チャネルは 8 バイトになります。

*4 外部サンプルで測定しているときは、本領域は不定の値となります。

マルチサンプルレートまたは低速サンプルレートでの注意事項

フリーランモードにおいて、マルチサンプルレートまたは低速サンプルレートで波形の取り込みをしている際に、ラッチ間で取得できる点数が、波形の取り込み中は一定の点数（16 点の倍数）となるようにデータサイズが調整されます。調整した結果、0 点となる場合は、次のラッチで取得できるデータに含まれます。

タイムスタンプ形式のデータについて

解析メニューで、電力解析、高調波解析、GPS 位置情報を有効に設定した場合、これらのチャネルはタイムスタンプ形式でデータが保存されます。タイムスタンプ形式のデータは、各アイテムの演算結果と、その演算を行った時刻情報が必ず組で保存されます。データはすべて LittleEndian 形式となります。

	電力解析		高調波解析		GPS 位置情報
チャネル	RMath13	RMath14	RMath15	RMath16	RMath1
アイテムのサブチャネル	1 ～ 62		1 ～ 62		1 ～ 6
	32bit 浮動小数点型		32bit 浮動小数点型		32bit 整数型
時刻情報のサブチャネル	63		63		7
	64bit 時刻形式（下記参照）				

時刻情報のサブチャネルは以下の形式で記録されます。

4 バイトのデータ	Unix Time（1970/1/1 を 0 とした値）	（1）の枠線部分
4 バイトのデータ	Tick（4Byte、nsec 単位（0 ～ 999999999））	（2）の枠線部分

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	EA	78	CC	61	28	97	A7	25	EA	78	CC	61	28	C4	D8	26
00000010	EA	78	CC	61	38	18	0A	28	EA	78	CC	61	38	45	3B	29
00000020	EA	78	CC	61	28	EB	6C	2A	EA	78	CC	61	38	9F	9D	2B
	(1)				(2)											

なお、外部サンプルで波形取り込みをした場合は、時刻情報ではなく、サンプルカウントが保存されます。

	サンプルカウント（最初のデータが 0 となる 64 ビットカウンタ値）		
8 バイトのデータ	4 バイトのデータ	サンプルカウント（上位 4 バイト）	（1）の枠線部分
	4 バイトのデータ	サンプルカウント（下位 4 バイト）	（2）の枠線部分

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	00	14	00	00	00	00	00	00	00	15	00	00	00
00000010	00	00	00	00	A4	01	00	00	00	00	00	00	A5	01	00	00
00000020	00	00	00	00	6C	02	00	00	00	00	00	00	6D	02	00	00
	(1)				(2)											

* サンプルカウントは単純な 64 ビットの整数値ではなく、上位／下位に分割された値となります。それぞれの値は LittleEndian 形式となります。

トリガモードについて

本 API と DL950/SL2000 を使用したトリガモードは以下のような仕組みで動作します。

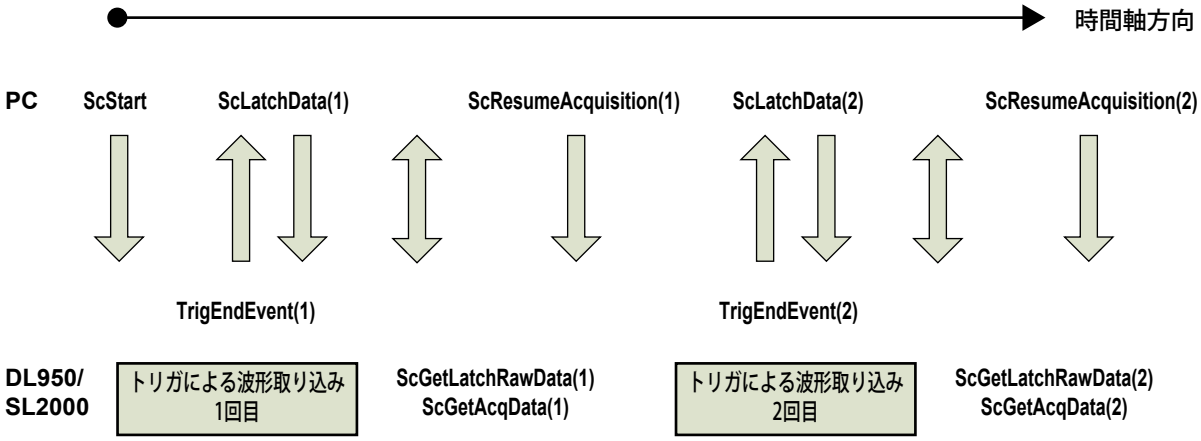
DL950/SL2000 は波形の取り込み開始（ScStart）を受け付けてから波形の取り込み停止（ScStop）を受け付けるまでトリガによる波形取得を行います。波形データは、本体のアクイジションメモリーに保存されます。（設定により保存可能なヒストリ数は変わります）
波形の取り込み中に、API からはラッチ（ScLatchData）および波形取得（ScGetAcqDataLength/ScGetAcqData）を繰り返し実行します。
トリガモードには、同期モードと非同期モードの 2 つのモードがあります。

同期モードについて

同期モードは、1 回の波形取り込みごとに DL950/SL2000 が PC からの応答を待ってから次の波形取り込みを行います。同期モードでは波形取り込みを行ったあとで確実に波形データを PC 側に転送したい場合に使用します。
DL950/SL2000 での 1 回の波形取り込みが終了したことを判断する方法は 2 つあり、一つ目はトリガエンドイベント（ScAddEventListener（C++）、ScAddCallBack（C#））を待つような実装を行います。二つ目はプログラムで定期的にアクイジション番号（ScGetLatchAcqCount）を監視して値が更新されていれば波形取り込みが終了していると判断します。
※ ScLatchData を実施してから ScGetLatchAcqCount を実行してください。

以下にトリガエンドイベントを用いたトリガによる波形取り込みの流れを示します。

トリガ同期モードの流れ

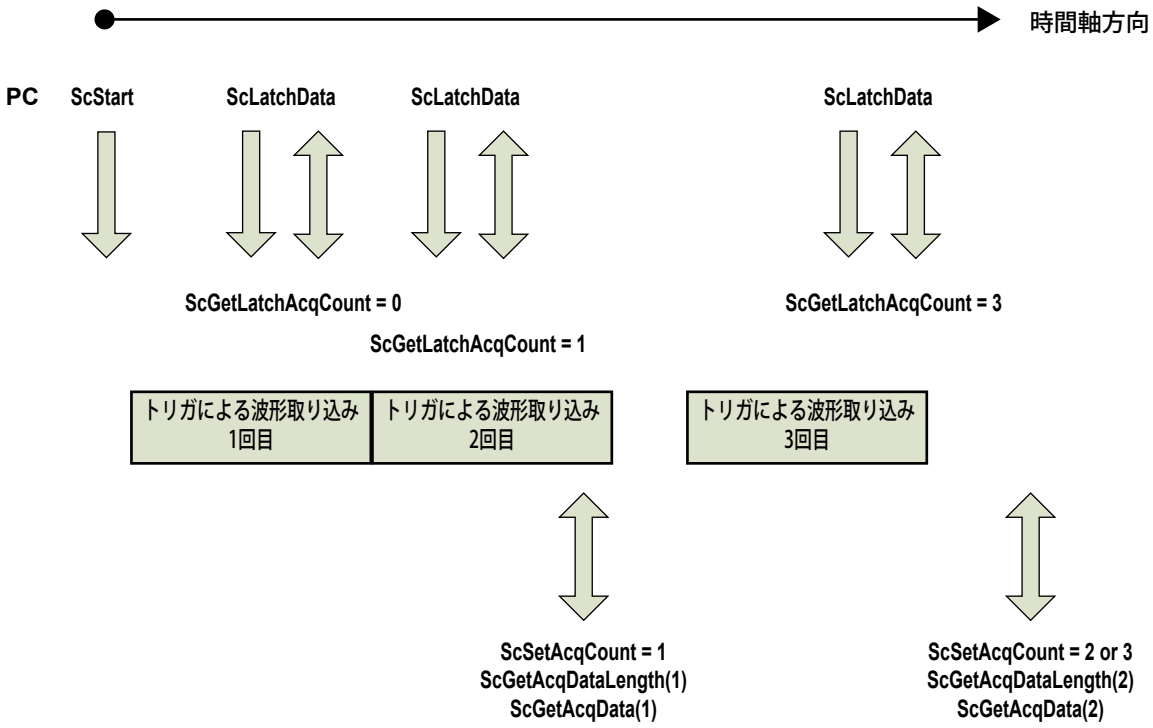


非同期モードについて

非同期モードは、DL950/SL2000 は PC 側のコマンド制御とは関係なく波形取り込みを続けます。
PC 側は定期的に ScLatchData と ScGetLatchAcqCount を実行してアクイジション番号の監視を行い、アクイジション番号が更新されていることを確認できたら、ScSetAcqCount で PC 側に転送したアクイジション番号を設定して波形データの転送を行います。
非同期モードは、主にトリガによる波形取り込みの間隔（デッドタイム）を短くしたい場合に使用します。
非同期モードの場合、PC からの波形取得に関係なく DL950/SL2000 は波形取り込みを繰り返し実行します。測定時間が短い（T/Div の設定が短い）設定の場合、PC で古いアクイジション番号を設定して波形取得を行う場合に、DL950/SL2000 の波形取り込みよりデータが上書きされる場合があります。
（DL950/SL2000 のヒストリ数に大きく依存します。ヒストリ数については、DL950 スコープコーダ /SL2000 高速データアクイジションユニット ユーザーズマニュアル [操作編]（IM DL950-02JA）の付録を参照してください）。この場合、PC で波形取得（ScGetAcqData）の命令を実行しても波形データの取得はできません。有効なデータ点数 0 となります。

以下に非同期モードによる波形取り込みの流れを示します。

トリガ非同期モードの流れ



外部サンプルによる波形取り込みについて

トリガモードで外部サンプルにて波形取り込みを行った場合、ScGetAcqData で取得できる timeSec および timeTick には時刻データはなく、サンプルカウントが以下の形式で格納されます。（タイムスタンプ形式の場合とは取り扱いが異なります。タイムスタンプ形式の詳細はフリーランモード側の説明を参照してください。）

8 バイトのデータ	サンプルカウント（波形取り込み開始した最初のデータが 0 となる 64 ビットカウンタ値）	
	4 バイトのデータ（timeSec）	サンプルカウント（下位 4 バイト）
	4 バイトのデータ（timeTick）	サンプルカウント（上位 4 バイト）

- サンプルカウントは単純な 64 ビットの整数値ではなく、上位／下位に分割された値となります。それぞれの値は LittleEndian 形式となります。
- サンプルカウントは波形取り込みを開始した最初のデータが 0 となります。トリガモードの場合、本 API で取得した最初のデータが 0 とは限りません。（トリガが成立し 1 回の取り込みが終了するまでサンプルカウントは加算し続けます。そのためデータ開始点のサンプルカウントは、終了した時点のサンプルカウントからレコード長分引いた値になります）

また、タイムスタンプ形式のチャネルの場合、演算区間ごとに値が出力されるためタイムスタンプ形式の時刻情報のサブチャネルに含まれるサンプルカウントは、通常のチャネルの ScGetAcqData で取得できる範囲と異なる場合があります。
タイムスタンプ形式（電力解析）の演算区間についての詳細については DL950 スコープコーダ /SL2000 高速データアクイジションユニット ユーザーズマニュアル [機能編]（IM DL950-01JA）の付録を参照してください。

5.2 フラッシュアクイジションデータアクセスライブラリ機能

DL950/SL2000 でアクイジションされている波形データの中で一部データの取り扱いに関して注意が必要となります。以下の内容に従ってデータを解析を行うようにしてください。

タイムスタンプ形式のデータについて

解析メニューで、電力解析、高調波解析、GPS 位置情報を有効に設定した場合、これらのチャネルはタイムスタンプ形式でデータが保存されます。タイムスタンプ形式のデータは、各アイテムの演算結果と、その演算を行った時刻情報が必ず組で保存されます。データはすべて LittleEndian 形式となります。

タイムスタンプ形式（電力解析）の演算区間についての詳細については、DL950 スコープコーダ /SL2000 高速データアクイジションユニット ユーザーズマニュアル [機能編]（IMDL950-01JA）の付録を参照してください。

	電力解析		高調波解析		GPS 位置情報
チャネル	RMath13	RMath14	RMath15	RMath16	RMath1
アイテムのサブチャネル	1 ～ 62		1 ～ 62		1 ～ 6
	32bit 浮動小数点型		32bit 浮動小数点型		32bit 整数型
時刻情報のサブチャネル	63		63		7
	64bit 時刻形式（下記参照）				

時刻情報のサブチャネルは以下の形式で記録されます。

4 バイトのデータ	Unix Time（1970/1/1 を 0 とした値）	（1）の枠線部分
4 バイトのデータ	Tick（4Byte、nsec 単位（0 ～ 999999999））	（2）の枠線部分

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	EA	78	CC	61	28	97	A7	25	EA	78	CC	61	28	C4	D8	26
00000010	EA	78	CC	61	38	18	0A	28	EA	78	CC	61	38	45	3B	29
00000020	EA	78	CC	61	28	EB	6C	2A	EA	78	CC	61	38	9F	9D	2B
					(1)				(2)							

なお、外部サンプルで波形取り込みをした場合は、時刻情報ではなく、サンプルカウントが保存されます。

	サンプルカウント（最初のデータが 0 となる 64 ビットカウンタ値）		
8 バイトのデータ	4 バイトのデータ	サンプルカウント（上位 4 バイト）	（1）の枠線部分
	4 バイトのデータ	サンプルカウント（下位 4 バイト）	（2）の枠線部分

ADDRESS	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	00	00	00	14	00	00	00	00	00	00	00	15	00	00	00
00000010	00	00	00	00	A4	01	00	00	00	00	00	00	A5	01	00	00
00000020	00	00	00	00	6C	02	00	00	00	00	00	00	6D	02	00	00
					(1)				(2)							

* サンプルカウントは単純な 64 ビットの整数値ではなく、上位／下位に分割された値となります。それぞれの値は LittleEndian 形式となります。

5.3 通信コマンドの使い方

通信コマンドは、以下の通信コマンド制御関数を使って送受信します。

API Name	Function	Page
ScSetControl	通信コマンドを送信する	4-7
ScGetControl	通信コマンドの応答を受信する	4-8
ScGetBinaryData	バイナリデータを受信する	4-9
ScQueryMessage	通信コマンドを送信しその応答を受信する	4-10

通信コマンド制御関数の詳細は 4.1 節「共通 API」を、コマンドの詳細は DL950 スコープコーダ /SL2000 高速データアクイジションユニット 通信インタフェース ユーザーズマニュアル（IM DL950-17）をご覧ください。

5.4 スコープコーダ SDK への移行

DL950ACQAPI もしくは DL950FACQAPI からスコープコーダ SDK へ移行する場合は、以下のようにプログラムを変更するとスムーズに移行できます。

C++ の場合
参照していた API を削除後、ライブラリに「ScSDK.lib」もしくは「ScSDK64.lib」を追加し、さらにヘッダファイルを「ScSDK.h」に変更すると ScSDK に移行できます。

C# の場合
参照していた API を削除後、インクルードに「ScSDKNet」を追加し、using を「using ScSDK = ScSDKNet.ScSDK;」と変更することで ScSDK に移行できます。

VB の場合
参照していた API を削除後、インクルードに「ScSDKNet」を追加し、Imports を「Imports ScSDK = ScSDKNet.ScSDK」と変更することで ScSDK に移行できます。

5.5 SL1000 コントロール API（SxAPI）との比較

以下は、SL1000 コントロール API と同じ機能を有するスコープコーダ SDK 関数の一覧になります。

関数概要	SL1000 コントロール API	スコープコーダ SDK
初期化	SxInit	ScInit
終了	SxExit	ScExit
コマンド送信	SxSetControl	ScSetControl
コマンド送受信	SxGetControl	ScGetControl
		ScQueryMessage
コマンド送受信	SxGetControlBinary	ScGetBinaryData
イベントハンドラ生成・通知許可	SxCreateEvent	ScAddEventListener
		ScAddCallback
イベントハンドラ削除	SxDeleteEvent	ScRemoveEventListener
		ScRemoveCallback
測定モードの設定	SxSetAcqMode	ScSetMeasuringMode
サンプル周波数の設定	SxSetSamplingRate	ScSetSamplingRate
サンプル周波数の問い合わせ	SxGetSamplingRate	ScGetSamplingRate
		ScGetChannelSamplingRate
測定開始	SxAcqStart	ScStart
		ScStartEx
測定停止	SxAcqStop	ScStop
		ScStopEx
ラッチ実行	SxAcqLatch	ScLatchData
		ScLatchDataEx
マニュアルトリガ発生	SxExecManualTrig	ScResumeAcquisition
収集データ情報の取得	SxGetChannelInfo	ScGetChannelGain
		ScGetChannelOffset
ラッチ区間サンプル点数の取得	SxGetLatchLength	ScGetAcqDataLength
最新アクイジション番号の取得	SxGetLatestAcqNo	ScGetLatchAcqCount
波形データの取得	SxGetAcqData	ScGetLatchRawData
		ScGetChAcqData
データ収集時刻の取得	SxGetAcqTime	ScGetTriggerTime
設定情報のセーブ	SxSaveSetup	ScSaveSetup
設定情報のロード	SxLoadSetup	ScLoadSetup
カレントドライブの設定	SxFileSetCurrentDrive	ScSetCurrentDrive
カレントドライブの問い合わせ	SxFileGetCurrentDrive	ScGetCurrentDrive
カレントディレクトリの設定	SxFileChDir	ScSetCurrentDirectory
カレントディレクトリの問い合わせ	SxFileCwDir	ScGetCurrentDirectory
ファイル数の取得	SxFileGetFileNum	ScGetFileNum
ファイル情報の取得	SxFileGetFileInfo	ScGetFileInfo
ファイルの削除	SxFileDelete	ScDeleteFile
ファイルの取得	SxFileGet	ScDownloadFile
ファイルの作成	SxFilePut	ScUploadFile

5.6 サンプルプログラム

本ソフトウェアは、以下のサンプルプログラムを同梱しています。

フォルダ名		内容
file		ファイル操作・転送機能（file）
flashacquition		フラッシュアクイジションデータアクセス
freerun	MultiUnit	複数台同期用データアクイジションフリーラン
	SingleUnit	データアクイジションフリーラン
reopen		測定器再接続及び測定モード設定
trigger	MultiUnit	複数台同期用データアクイジショントリガ
	SingleUnit	データアクイジショントリガ

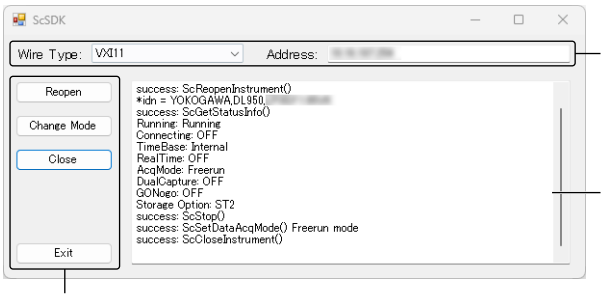
それぞれのサンプルプログラムは、C++、C#、VBNet に対応しています。

フォルダ名	内容
ScSDKNetSample	サンプルプログラム（C#）
ScSDKSample	サンプルプログラム（C++）
ScSDKVBNetSample	サンプルプログラム（VBNet）

ご使用の環境に合わせて、これらのサンプルプログラムを参考に、本書に記載された API を利用してください。

測定器再接続及び測定モード設定（reopen）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Address」テキストボックス
  接続する機器のアドレスを入力するテキストボックスです。

操作ボタン

- 「Reopen」ボタン
  選択した Wire Type と入力したアドレスに一致する測定中の機器に接続を行います。
  さらにステータス情報を取得し、測定中の場合は測定を停止します。
  使用する API：ScInit、ScReopenInstrument、ScGetStatusInfo、ScStop
- 「Change Mode」ボタン
  接続中の機器をフリーランモードに変更します。
  使用する API：ScSetMeasuringMode
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrument
- 「Exit」ボタン
  アプリケーションを終了します。
  使用する API：ScExit

ログ画面

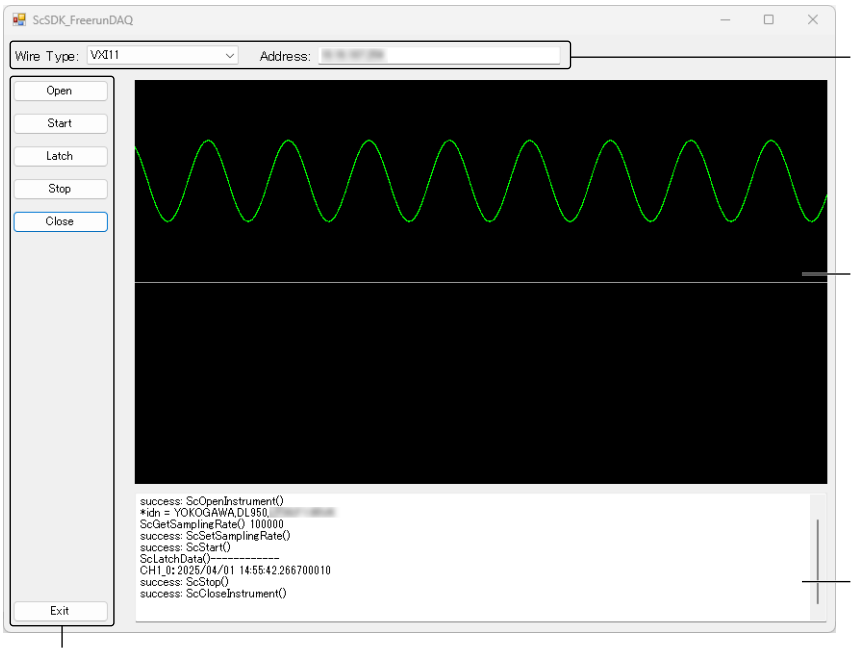
接続状況やデータの詳細を表示します。

使用方法

1. 「Wire Type」コンボボックスで回線種別を、「Address」テキストボックスで接続先を指定します。
2. 「Reopen」ボタンを押し、接続を開始します。接続した機器のステータス情報を取得し、測定中だった場合は測定を停止します。
3. 「Change Mode」ボタンを押し、測定モードをフリーランモードに変更します。
4. 「Close」ボタンで接続を終了します。
5. 「Exit」ボタンでアプリケーションを終了します。

データアクイジションフリーラン（freerun SingleUnit）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Address」テキストボックス
  接続する機器のアドレスを入力するテキストボックスです。

操作ボタン

- 「Open」ボタン
  選択した Wire Type と入力したアドレスに一致する機器に接続を行います。
  使用する API：ScInit、ScOpenInstrument、ScGetSamplingRate
- 「Start」ボタン
  測定を開始します。
  使用する API：ScStart
- 「Latch」ボタン
  ラッチを行います。
  使用する API：ScLatchData、ScGetLatchRawData、ScGetChAcqData、ScGetChannelGain、ScGetChannelOffset、ScGetChannelBits、ScGetChannelType、ScGetChannelScale
- 「Stop」ボタン
  測定を終了します。
  使用する API：ScStop
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrument
- 「Exit」ボタン
  アプリケーションを終了します。
  使用する API：ScExit

波形表示画面

データを取得し波形を表示します。デフォルトでは CH1 のみのデータを取得し、表示します。

ログ画面

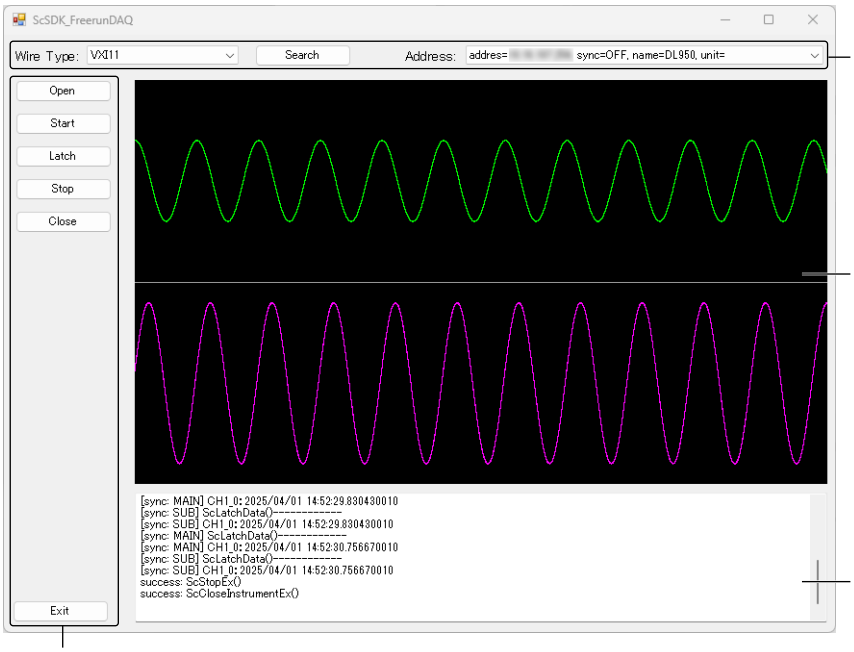
接続状況やデータの詳細を表示します。

使用方法

1.「Wire Type」コンボボックスで回線種別を、「Address」テキストボックスで接続先を指定します。
2.「Open」ボタンを押し、接続を開始します。
3.「Start」ボタンを押し、測定を開始します。
4.「Latch」ボタンを押し、ラッチ間のデータを取得します。取得したデータは波形表示画面に表示されます。
5.「Stop」ボタンで測定を終了します。測定を再び開始したい場合は、「Start」ボタンを押します。
6.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
7.「Exit」ボタンでアプリケーションを終了します。

複数台同期用データアクイジションフリーラン（freerun MultiUnit）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Search」ボタン
  選択した回線種別で接続可能な測定器のリストを取得します。
  使用する API：ScInit、ScSearchDevices
- 「Address」コンボボックス
  接続可能な測定器のリストが表示されます。

操作ボタン

- 「Open」ボタン
  選択した Wire Type と入力したアドレスに一致する機器に接続を行います。
  使用する API：ScOpenInstrumentEx、ScGetSamplingRate
- 「Start」ボタン
  測定を開始します。
  使用する API：ScStartEx
- 「Latch」ボタン
  ラッチを行います。
  使用する API：ScLatchDataEx、ScGetLatchRawData、ScGetChAcqData、ScGetChannelGain、ScGetChannelOffset、ScGetChannelBits、ScGetChannelType、ScGetChannelScale
- 「Stop」ボタン
  測定を終了します。
  使用する API：ScStopEx
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrumentEx

・「Exit」ボタン
　アプリケーションを終了します。
　使用する API：ScExit

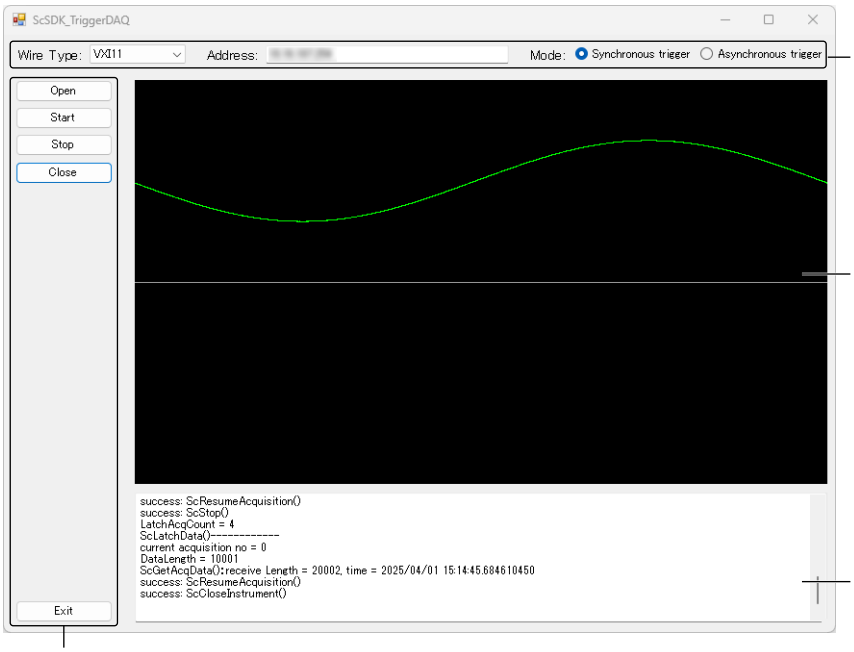
波形表示画面
　データを取得し波形を表示します。デフォルトでは CH1 のみのデータを取得し、表示します。

ログ画面
　接続状況やデータの詳細を表示します。

使用方法
1.「Wire Type」コンボボックスで回線種別を選択します。
2.「Search」ボタンを押し、接続可能な測定器を取得します。
3.「Address」コンボボックスで接続先を選択します。
4.「Open」ボタンを押し、接続を開始します。
5.「Start」ボタンを押し、測定を開始します。
6.「Latch」ボタンを押し、ラッチ間のデータを取得します。取得したデータは波形表示画面に表示されます。
7.「Stop」ボタンで測定を終了します。再び測定を開始したい場合は、「Start」ボタンを押します。
8.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
9.「Exit」ボタンでアプリケーションを終了します。

データアクイジショントリガ（trigger SingleUnit）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Address」テキストボックス
  接続する機器のアドレスを入力するテキストボックスです。
- 「Mode」ラジオボタン
  同期モードか非同期モードか選択できます。

操作ボタン

- 「Open」ボタン
  選択した Wire Type と入力したアドレスに一致する機器に接続を行います。
  使用する API：ScInit、ScOpenInstrument、ScAddCallback（C#、VBNet の場合）、ScAddEventListener（C++ の場合）、ScGetSamplingRate、ScSetTriggerTimeout、ScGetTriggerTimeout
- 「Start」ボタン
  測定を開始します。
  使用する API：ScStart
- 「Stop」ボタン
  測定を終了します。
  使用する API：ScStop
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrument、ScRemoveCallback（C#、VBNet の場合）、ScRemoveEventListener（C++ の場合）
- 「Exit」ボタン
  アプリケーションを終了します。
  使用する API：ScExit

波形表示画面

DL950/SL2000 からのトリガイベントを受け取ると、波形データを取得し波形を表示します。デフォルトでは CH1 のみのデータを取得し、表示します。

使用する API：ScLatchData、ScGetLatchAcqCount、ScSetAcqCount、ScGetAcqCount、ScGetAcqData、ScGetAcqDataLength、ScGetChannelGain、ScGetChannelOffset、ScGetChannelBits、ScGetChannelType、ScGetChannelScale、ScResumeAcquisition

ログ画面

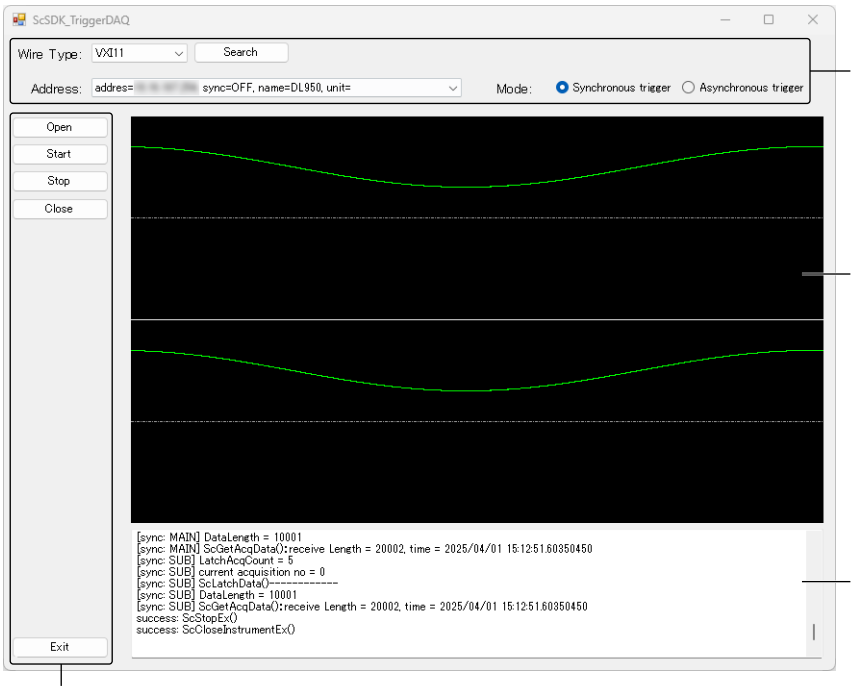
接続状況やデータの詳細を表示します。

使用方法

1.「Wire Type」コンボボックスで回線種別を、「Address」テキストボックスで接続先を指定します。
2.「Mode」ラジオボタンで同期モードか非同期モードかを選択します。
3.「Open」ボタンを押し、接続を開始します。
4.「Start」ボタンを押し、測定を開始します。開始後、同期モードの場合トリガイベントが発生すると、非同期モードの場合 100 ミリ秒でラッチを行い、波形表示画面に波形を表示します。
5.「Stop」ボタンで測定を終了します。再び測定を開始したい場合は、「Start」ボタンを押します。
6.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
7.「Exit」ボタンでアプリケーションを終了します。

複数台同期用データアクイジショントリガ（trigger MultiUnit）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Search」ボタン
  選択した回線種別で接続可能な測定器のリストを取得します。
  使用する API：ScInit、ScSearchDevices
- 「Address」コンボボックス
  接続可能な測定器のリストが表示されます
- 「Mode」ラジオボタン
  同期モードか非同期モードか選択できます。

操作ボタン

- 「Open」ボタン
  選択した機器に接続を行います。
  使用する API：ScOpenInstrumentEx、ScAddCallback（C#、VBNet の場合）、ScAddEventListener（C++ の場合）、ScGetSamplingRate、ScSetTriggerTimeout、ScGetTriggerTimeout
- 「Start」ボタン
  測定を開始します。
  使用する API：ScStartEx
- 「Stop」ボタン
  測定を終了します。
  使用する API：ScStopEx

・「Close」ボタン
接続を切断します。
使用する API：ScCloseInstrumentEx、ScRemoveCallback（C#、VBNet の場合）、ScRemoveEventListener（C++ の場合）
・「Exit」ボタン
アプリケーションを終了します。
使用する API：ScExit

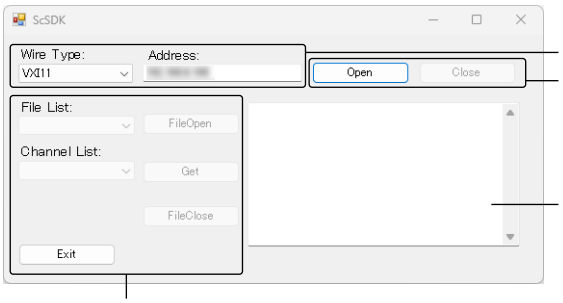
波形表示画面
DL950/SL2000 からのトリガイベントを受け取ると、波形データを取得し波形を表示します。デフォルトでは CH1 のみのデータを取得し、表示します。
使用する API：ScLatchDataEx、ScLatchData、ScGetLatchAcqCount、ScSetAcqCount、ScGetAcqCount、ScGetAcqData、ScGetAcqDataLength、ScGetChannelGain、ScGetChannelOffset、ScGetChannelBits、ScGetChannelType、ScGetChannelScale、ScResumeAcquisition

ログ画面
接続状況やデータの詳細を表示します。

使用方法
1.「Wire Type」コンボボックスで回線種別を選択します。
2.「Search」ボタンを押し、接続可能な測定器を取得します。
3.「Address」コンボボックスで接続先を指定します。
4.「Mode」ラジオボタンで同期モードか非同期モードかを選択します。
5.「Open」ボタンを押し、接続を開始します。
6.「Start」ボタンを押し、測定を開始します。開始後、同期モードの場合トリガイベントが発生すると、非同期モードの場合 100 ミリ秒でラッチを行い、波形表示画面に波形を表示します。
7.「Stop」ボタンで、測定を終了します。再び測定を開始したい場合は、「Start」ボタンを押します。
8.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
9.「Exit」ボタンでアプリケーションを終了します。

フラッシュアクイジションデータアクセス（flashacquition）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Address」テキストボックス
  接続する機器のアドレスを入力するテキストボックスです。

接続開始及び終了ボタン

- 「Open」ボタン
  選択した Wire Type と入力したアドレスに一致する機器に接続を行います。
  使用する API：ScInit、ScOpenInstrument、ScGetFAcqCount、ScGetFAcqFileName
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrument

操作ボタン

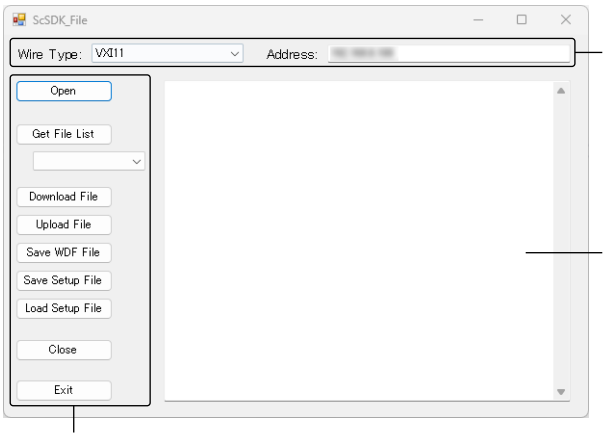
- 「File List」コンボボックス
  接続開始後、アクイジションファイル名がコンボボックスに保存されます。
- 「FileOpen」ボタン
  「File List」コンボボックスで選択したアクイジションファイルを開き、チャネルリストを「Channel List」コンボボックスに保存します。
  使用する API：ScOpenFAcqData、ScGetFAcqChannelCount、ScGetFAcqChannelNumber、ScSetFAcqDataSize、ScSet10GMode、ScGet10GMode
- 「Channel List」コンボボックス
  アクイジションファイルのチャネルリストが保存されます。
- 「Get」ボタン
  「Channel List」コンボボックスで選択したチャンネルデータを CSV ファイルとして保存します。
  使用する API：ScSetFAcqChannelNumber、ScGetFAcqChannelBits、ScGetFAcqDataLength、ScGetFAcqComment、ScGetFAcqChannelLabel、ScGetFAcqChannelUnit、ScGetFAcqChannelHResolution、ScGetFAcqChannelHoffset、ScGetFAcqTimeBase、ScGetFAcqStartTime、ScGetFAcqChannelGain、ScGetFAcqChannelOffset、ScGetFAcqData
- 「FileClose」ボタン
  アクイジションファイルを閉じます。
  使用する API：ScCloseFAcqData
- 「Exit」ボタン
  アプリケーションを終了します。
  使用する API：ScExit

ログ画面
接続状況やデータの詳細を表示します。

使用方法
1.「Wire Type」コンボボックスで回線種別を、「Address」テキストボックスで接続先を指定します。
2.「Open」ボタンを押し、接続を開始します。接続先のアクイジションファイルのリストが「File List」コンボボックスに表示されます。
3.「File List」コンボボックスでアクイジションファイルを選択し、「FileOpen」ボタンを押します。アクイジションファイルのチャネルリストが「Channel List」コンボボックスに表示されます。
4.「Channel List」コンボボックスでチャンネルを選択して「Get」ボタンを押し、チャネルデータを取得します。
5.「FileClose」ボタンで、アクイジションファイルを閉じます。
6.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
7.「Exit」ボタンでアプリケーションを終了します。

ファイル操作・転送機能（file）

初期画面



接続先の設定

- 「Wire Type」コンボボックス
  回線種別を USBTMC、VXI11、HiSLIP から選ぶことができます。
- 「Address」テキストボックス
  接続する機器のアドレスを入力するテキストボックスです。

操作ボタン

- 「Open」ボタン
  選択した Wire Type と入力したアドレスに一致する機器に接続を行います。
  使用する API：ScInit、ScOpenInstrument
- 「Get File List」ボタン
  カレントディレクトリのファイルのリストを取得し、コンボボックスに表示します。
  使用する API：ScGetFileList
- 「Download File」ボタン
  コンボボックスで選択したファイルを取得します。
  使用する API：ScDownloadFile
- 「Upload File」ボタン
  先ほど取得したファイルを機器に保存します。
  使用する API：ScUploadFile
- 「Save WDF File」ボタン
  画面に表示された波形を WDF ファイルで取得します。
  使用する API：ScSaveTriggerWDF
- 「Save Setup File」ボタン
  機器の設定を SET ファイルで取得します。
  使用する API：ScSaveSetup
- 「Load Setup File」ボタン
  先ほど取得した SET ファイルを機器の設定に反映させます。
  使用する API：ScLoadSetup
- 「Close」ボタン
  接続を切断します。
  使用する API：ScCloseInstrument
- 「Exit」ボタン
  アプリケーションを終了します。
  使用する API：ScExit

ログ画面

接続状況やデータの詳細を表示します。

使用方法

1.「Wire Type」コンボボックスで回線種別を、「Address」テキストボックスで接続先を指定します。
2.「Open」ボタンを押し、接続を開始します。
3.「Get File List」ボタンを押し、カレントディレクトリのファイルリストを取得します。取得したファイルリストはコンボボックスに表示されます。
4. コンボボックスでファイルを選択します。
5.「Download File」ボタンを押し、コンボボックスで選択したファイルを取得します。
6.「Upload File」ボタンを押し、先ほど取得したファイルを機器に保存します。
7.「Save WDF File」ボタンを押し、機器の画面に表示された波形を WDF ファイル形式で保存します。画面に波形がない場合エラーになります。
8.「Save Setup File」ボタンを押し、機器の設定を SET ファイル形式で取得します。
9.「Load Setup File」ボタンを押し、先ほど取得した SET ファイルを機器の設定に反映します。
10.「Close」ボタンで接続を終了します。再び接続したい場合は、「Open」ボタンを押します。
11.「Exit」ボタンでアプリケーションを終了します。