User's Manual

ScopeCorder SDK

YOKOGAWA
Yokogawa Test & Measurement Corporation

IM D165-01EN
1st Edition

This user's manual contains useful information about the precautions, functions, and API specifications of the ScopeCorder SDK.
To ensure correct use, please read this manual thoroughly before operation. Keep this manual in a safe place for quick reference.
For information about the handling precautions, functions, and operating procedures of the DL950/SL2000 series and the handling and operating procedures of Windows, see the relevant manuals.

Notes

- The contents of this manual are subject to change without prior notice as a result of improvements to the product's performance and functionality. Refer to our website to view our latest manuals.
- The figures given in this manual may differ from those that actually appear on your screen.
- Every effort has been made in the preparation of this manual to ensure the accuracy of its contents. However, should you have any questions or find any errors, please contact your nearest YOKOGAWA dealer.
- Copying or reproducing all or any part of the contents of this manual without the permission of YOKOGAWA is strictly prohibited.

Trademarks

- Microsoft, Windows, Windows 10, Windows 11, and Visual Studio are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe and Acrobat are either registered trademarks or trademarks of Adobe Inc.
- In this manual, the ® and TM symbols do not accompany their respective registered trademark or trademark names.
- Other company and product names are registered trademarks or trademarks of their respective holders.

Revisions

1st Edition: June 2025

1st Edition: June 2025 (YMI)

Notes on Usage

Handling Precautions

- This software is an API library for the DL950/SL2000 series. It cannot be used with other products.
- Check the version of this software and the firmware version of the DL950/SL2000 prior to use. This software is compatible with DL950/SL2000 firmware version 2.01 and later.
- For information on how to use the DL950/SL2000, refer to each manual.

Disclaimers

By downloading and installing this software, the customer agrees to all of the following disclaimers.

- Yokogawa bears no liability for any problems occurring as a result of downloading or installing this software.
- Yokogawa bears no responsibility for any damage caused directly or indirectly as a result of using this software.
- This software is provided free of charge, however no unlimited warranty against software defects exists, nor is any claim made that the product is free of all defects whatsoever. Also, Yokogawa is not always able to repair defects (“bugs”) in, or respond to questions or inquiries about this software.
- Yokogawa reserves all rights to this software, including but not limited to all property rights, ownership rights, and intellectual property rights.

Contents

Chapter 5 Appendix

1.1 Software Overview

Description

This software (ScopeCorder SDK) provides application programming interface (API) for waveform data acquisition in DL950/SL2000 series, transfer of data stored in flash acquisition memory to PC, file operation and file transfer functions.

Function

This software can be used to perform the following functions. For details, see “Detailed API Specifications.”

- Initializing the API
- Connecting and disconnecting from measuring instruments
- Setting parameters
- Getting waveform data
- Getting the measurement conditions of waveform data stored in the flash acquisition area
- Getting the waveform data stored in the flash acquisition area
- Getting the list of files stored in the instrument
- Operating and transferring files (instrument to PC and PC to instrument)

Note

For features not covered by this API (mainly channel (vertical-axis) settings), implement them using communication commands by referring to the *DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit Communication Interface User’s Manual*, IM DL950-17EN.

Software structure

This software consists of the following:

Folder name	File name	Description
	readme.txt	Version information of ScopeCorder SDK
dll	ScSDK.dll	API Main Unit
	ScSDK64.dll	API Library 64-bit Version
	ScSDKNet.dll	Free Run API Library for .NET
	tmctl.dll	Communication Library (7.0.0.0)
	tmctl64.dll	Communication Library 64-bit Version (7.0.0.0)
doc	IMD165-01EN_010.pdf	ScopeCorder SDK User’s Manual (this manual)
sample		Sample Programs For details, see section 5.6, “Sample Programs.”
vc	ScSDK.lib	API Import Library (C++ only)
	ScSDK64.lib	API Import Library 64-bit Version (C++ only)
	ScSDK.h	Function Declaration Header File (C++ only)

PC System Requirements

PC

A PC that meets the following conditions is required.

A PC running the English or Japanese version of Windows 10 (32 bit or 64 bit)

A PC running the English or Japanese version of Windows 11

Note that when waveforms are acquired in free run mode using this software, data is saved in a specified buffer. For the memory size required by the API, see “Required memory size” in section 5.1, “Free Run Mode.”

Development Environment

Visual Studio 2017 or later, .NET Framework 4.7 or later

System requirements for running user programs

The following environment may be necessary to perform waveform acquisition in free run mode using a program that you create with this software depending on your waveform acquisition conditions and connection type.

When using 10Gbit Ethernet connection

- CPU
  Desktop-Type PC
  Intel Core i7-1165G7 or better, quad core (8 threads) or better, 4.7 GHz or faster
- Memory
  16 GB or more
- SSD
  512 GB or more (M.2 slot SSD recommended, read/write performance 3 GB/s or better)

When using 1Gbit Ethernet or USB connection

- CPU
  Intel Core i5-10210U or better, quad core (8 threads) or better, 4.2 GHz or faster
- Memory
  8 GB or more
- SSD
  256 GB or more (read/write performance 400 MB/s or better)

USB driver

To use this software over a USB connection, you need a dedicated USB driver (YTUSB) or an IVI driver (VISA). You can download the latest USB driver from the following web page:

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

Run Setup.exe in the YTUSB folder. The installation wizard starts. For details on the installation procedure, see the manual (ReadMe_en.pdf) in the YTUSB folder.

DL950/SL2000 firmware version

This software can be used with DL950/SL2000 with firmware version 2.01 or later. Download the latest firmware from the our web page.

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

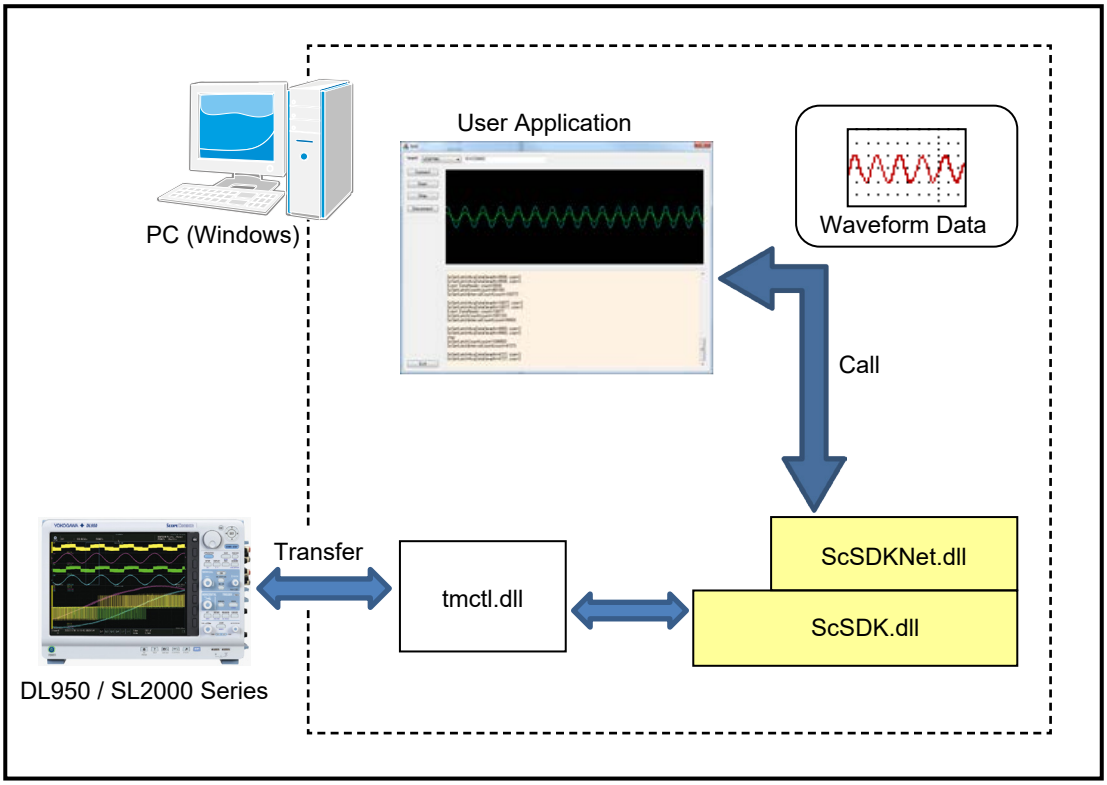
2.1 API Overview

The API is provided as a dynamic link library (DLL). The API can be used by linking user applications with this DLL.
The API provides the following three functions.

- Data Acquisition Function
- Flash acquisition data access library
- File operation and transfer feature

Data Acquisition Function

As shown in the following figure, the data acquisition function provides functions for obtaining waveform data being acquired by the instrument and setting measurement conditions to the application. (Free run measurement and trigger measurement)



The API's data acquisition function supports two acquisition modes: free run and trigger.

(1) Free run mode
Free run mode is used to acquire data from the start to the end of waveform acquisition.

Waveform acquisition specifications in free run mode
Maximum data rate 320 MB/s (10 MS/s×16ch) for 10Gbit Ethernet connection
Maximum data rate 6.4 MB/s (200 kS/s×16ch) for 1Gbit Ethernet/USB connection
Maximum waveform acquisition time 10 days (maximum operation time guaranteed for this API)*

* If data is sent from a DL950/SL2000 at the above data rate in measurement using multi-unit synchronization connection, the possibility of data transmission buffer overrun occurring will increase depending on the connection environment, the PC performance, and so on. As such, it is recommended that measurements be made with the total data rate of the multiple connected units set within the above range.

(2) Trigger mode

Trigger mode is used to acquire waveform using triggers. There are two trigger modes available with the API: (1) synchronous mode in which the DL950/SL2000 acquires waveforms synchronously with the PC and (2) asynchronous mode in which the DL950/SL2000 acquires waveforms asynchronously with the PC.

Note that the API does not support the following features.

- Waveform acquisition in roll mode (the DL950/SL2000 itself supports waveform acquisition in roll mode, but the API does not support waveform acquisition while the DL950 is acquiring waveforms in roll mode)
- DL950/SL2000 trigger mode set to Single N
- Waveform acquisition using dual capture
- Real-time recording (SSD and flash acquisition)
- Recorder mode

Trigger-based waveform acquisition specifications

Maximum waveform acquisition time	10 days (maximum operation time guaranteed for this API)*

When high-speed transmission mode using 10GbpsEthernet is enabled, the maximum record length that can be specified is as shown below due to the memory join limitation. For details on memory join, see the appendix in the DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit User's Manual (IM DL950-03EN).

Standard model: 250 M
/M1 Model: 1 G
/M2 Model: 2 G

Flash acquisition data access library

As shown in the following figure, the Flash Acquisition Data Access Library provides applications with a function for extracting flash acquisition waveform data stored in a DL950/SL2000 directly to a PC without loading the data into the instrument.

* Note that the acquisition memory is used as a temporary buffer when flash acquisition waveform data is extracted through the use of this API. Thus, data and history information in the acquisition memory that have not been saved to a storage device will be cleared. Waveform data stored in the flash acquisition area is not affected.

* This feature is available only when the /ST2 option is installed.

File operation and transfer feature

The file operation and transfer feature provides applications with features related to the acquisition, transmission, and deletion of DL950/SL2000 measurement data and settings.

2.2 Overview of API Functions

This section provides an overview of the API functions.

Initialization and termination

The API functions for initialization and termination are as follows.

API Name	Function	Page
ScInit	Initialize the API	4-1
ScExit	Close the API	4-1

Connection and disconnection

The API functions for connecting and disconnecting from the measurement instrument are as follows.

API Name	Function	Page
ScOpenInstrument	Open an instrument and get the API handle	4-2
ScReopenInstrument	Reopen the instrument	4-3
ScCloseInstrument	Close the instrument	4-4
ScOpenInstrumentEx	Connect to the instruments in multi-unit synchronization and get the list of connection handles	4-4
ScCloseInstrumentEx	Disconnect from the instruments using the connection handle list	4-7
ScSearchDevices	Get the list of connectable instruments	4-12

Getting or setting waveform acquisition conditions

The API functions for getting and setting waveform acquisition conditions are as follows.

API Name	Function	Page
ScSetControl	Send a command to the instrument	4-7
ScGetControl	Receive a command response from the instrument	4-8
ScQueryMessage	Send a command and receive a response	4-10
ScGetBinaryData	Receive binary data	4-9
ScSetSamplingRate	Set the sampling rate	4-27
ScGetSamplingRate	Get the sampling rate	4-28
ScGetChannelSamplingRate	Get the channel sampling rate	4-28
ScSet10GMode	Sets the 10G high-speed transmission mode	4-11
ScGet10GMode	Gets the 10G high-speed transmission mode	4-12
ScStart	Start waveform acquisition	4-15
ScStop	Stop waveform acquisition	4-15
ScGetStatusInfo	Get the status information of the instrument	4-13
ScTmcSetTimeout	Set the TMCTL timeout period	4-13

Getting trigger-based waveform acquisition information

The API functions for getting trigger-based waveform acquisition information are as follows.

API Name	Function	Page
ScSetMeasuringMode	Set the data acquisition mode	4-14
ScGetLatchAcqCount	Get the latest acquisition count at the latch point	4-23
ScGetAcqCount	Get the acquisition count for acquiring data	4-24
ScSetAcqCount	Set the acquisition count for acquiring data	4-24
ScGetTriggerTime	Get the trigger time for the specified acquisition count	4-25
ScResumeAcquisition	Resume waveform acquisition in synchronous mode	4-25
ScSetTriggerTimeout	Set the timeout value on the DL950/SL2000 in synchronous mode	4-26
ScGetTriggerTimeout	Get the timeout value on the DL950/SL2000 in synchronous mode	4-26

Get waveform data

The API functions for getting waveform data in free run mode are as follows.

API Name	Function	Page
ScLatchData	Latch the waveform acquisition information	4-16
ScGetLatchRawData	Get waveform data after latching	4-18
ScGetChAcqData	Get waveform data information of a specified channel from the block data obtained using ScGetLatchRawData	4-19
ScGetFreeRunDataLength	Get the number of valid points according to the time base setting in free run mode.	4-23

The API functions for getting waveform data in trigger mode are as follows.

API Name	Function	Page
ScLatchData	Latch the waveform acquisition information	4-16
ScGetAcqData	Get the measurement data for the specified acquisition count	4-21
ScGetAcqDataLength	Get the data length for the specified acquisition count	4-22

Converting waveform data

The API functions for converting waveform data into physical values are as follows.

API Name	Function	Page
ScGetChannelBits	Get the data bit count of the channel	4-29
ScGetChannelGain	Get the gain value of the channel (used to convert waveform data into actual data)	4-29
ScGetChannelOffset	Get the offset value of the channel (used to convert waveform data into actual data)	4-30
ScGetChannelScale	Get the upper and lower limits of the channel display scale	4-30
ScGetChannelType	Get the type of channel waveform data	4-31

Event listener and callback functions

The event listener and callback API functions are as follows.

API Name	Function	Page
ScAddEventListener	Add an event listener (C++ only)	4-31
ScRemoveEventListener	Delete the event listener (C++ only)	4-32
ScAddCallback	Add a call back method (C# only)	4-33
ScRemoveCallback	Delete the call back method (C# only)	4-33

Getting flash acquisition waveform data information

The API functions for getting flash acquisition waveform data information are as follows.

API Name	Function	Page
ScGetFAcqCount	Getting the number of flash acquisition waveform data files stored in the instrument	4-34
ScGetFAcqFileName	Get the name of a flash acquisition waveform data file stored in the instrument	4-34
ScOpenFAcqData	Open a waveform data file for data transmission	4-35
ScCloseFAcqData	Close the opened waveform data file	4-35
ScClearAcqMemory	Clear the waveform data in the acquisition memory	4-36
ScGetFAcqStartTime	Get the measurement start time of the opened waveform data file	4-36
ScGetFAcqTimeBase	Get the time base setting of the opened waveform data file	4-37
ScGetFAcqComment	Get the comment for the opened waveform data file	4-37
ScGetFAcqChannelCount	Get the total number of channels stored in the opened waveform data file	4-38
ScGetFAcqChannelNumber	Get the channel numbers stored in the opened waveform data file	4-38

Getting the channel information stored in a waveform data file

The API functions for getting the channel information stored in the opened waveform data file are as follows.

API Name	Function	Page
ScGetFAcqChannelBits	Get the specified channel's number of bits per data point	4-39
ScGetFAcqChannelGain	Get the specified channel's gain	4-40
ScGetFAcqChannelHOffset	Get the specified channel's offset time from the measurement start time	4-40
ScGetFAcqChannelHResolution	Get the specified channel's sampling interval	4-41
ScGetFAcqChannelLabel	Get the specified channel's label name	4-42
ScGetFAcqChannelLogicBits	Get the effective number of bits when the specified channel is a logic channel	4-42
ScGetFAcqChannelLogicLabel	Get the label name of each bit when the specified channel is a logic channel	4-43
ScGetFAcqChannelOffset	Get the specified channel's offset value	4-43
ScGetFAcqChannelSign	Get the specified channel's sign information.	4-44
ScGetFAcqChannelType	Get the specified channel's data type	4-45
ScGetFAcqChannelUnit	Get the specified channel's unit string	4-45

Get waveform data

The API functions for getting waveform data are as follows.

API Name	Function	Page
ScSetFAcqChannelNumber	Set the channel number you want to get the waveform from	4-46
ScGetFAcqDataLength	Get the number of data points in the waveform to be acquired	4-46
ScGetFAcqData	Get the specified channel's data	4-47
ScSetFAcqDataSize	Set the maximum data size to send at one time	4-48
ScSet10GMode	Sets the 10Gbps high-speed transmission mode	4-11
ScGet10GMode	Gets the 10Gbps high-speed transmission mode	4-12

Operating and transferring files

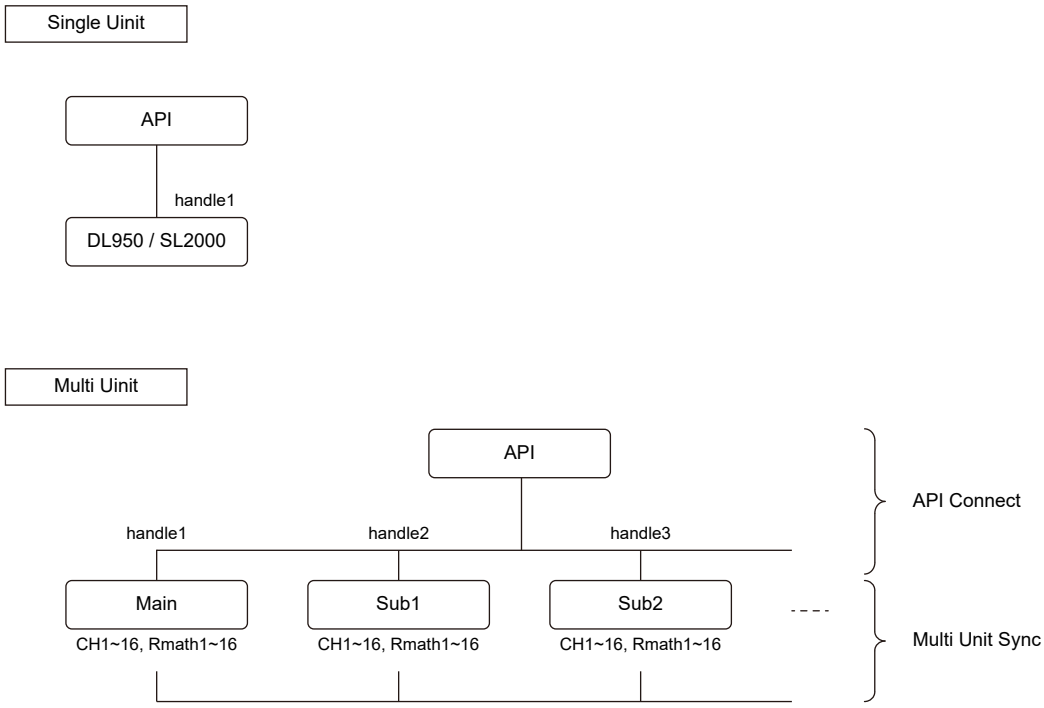
The API functions for operating and transferring files are as follows.

API Name	Function	Page
ScSetCurrentDrive	Set the current drive	4-49
ScGetCurrentDrive	Get the current drive	4-49
ScSetCurrentDirectory	Set the current directory	4-50
ScGetCurrentDirectory	Get the current directory	4-50
ScGetFileNum	Get the number of files	4-51
ScGetFileInfo	Get file information	4-51
ScDeleteFile	Delete a file	4-52
ScDownloadFile	Get the instrument file	4-52
ScUploadFile	Save a file in the instrument	4-53
ScSaveTriggerWDF	Get a trigger waveform as a WDF file	4-53
ScSaveFreeRunWDF	Get a free run waveform as a WDF file	4-54
ScSaveSetup	Get the instrument settings	4-54
ScLoadSetup	Apply the setup file settings to the instrument	4-55
ScGetFileList	Get the file list	4-55
ScGetFileInfoList	Get the file information list	4-57

2.3 Basic Flow of Using the API

Handle

Each API function is used through a handle. First, a handle is created when an instrument is opened. Then, the target instrument is accessed by passing the handle as an API parameter.
When connecting to instruments in multi-unit synchronization connection, a handle is created for each instrument to be accessed.



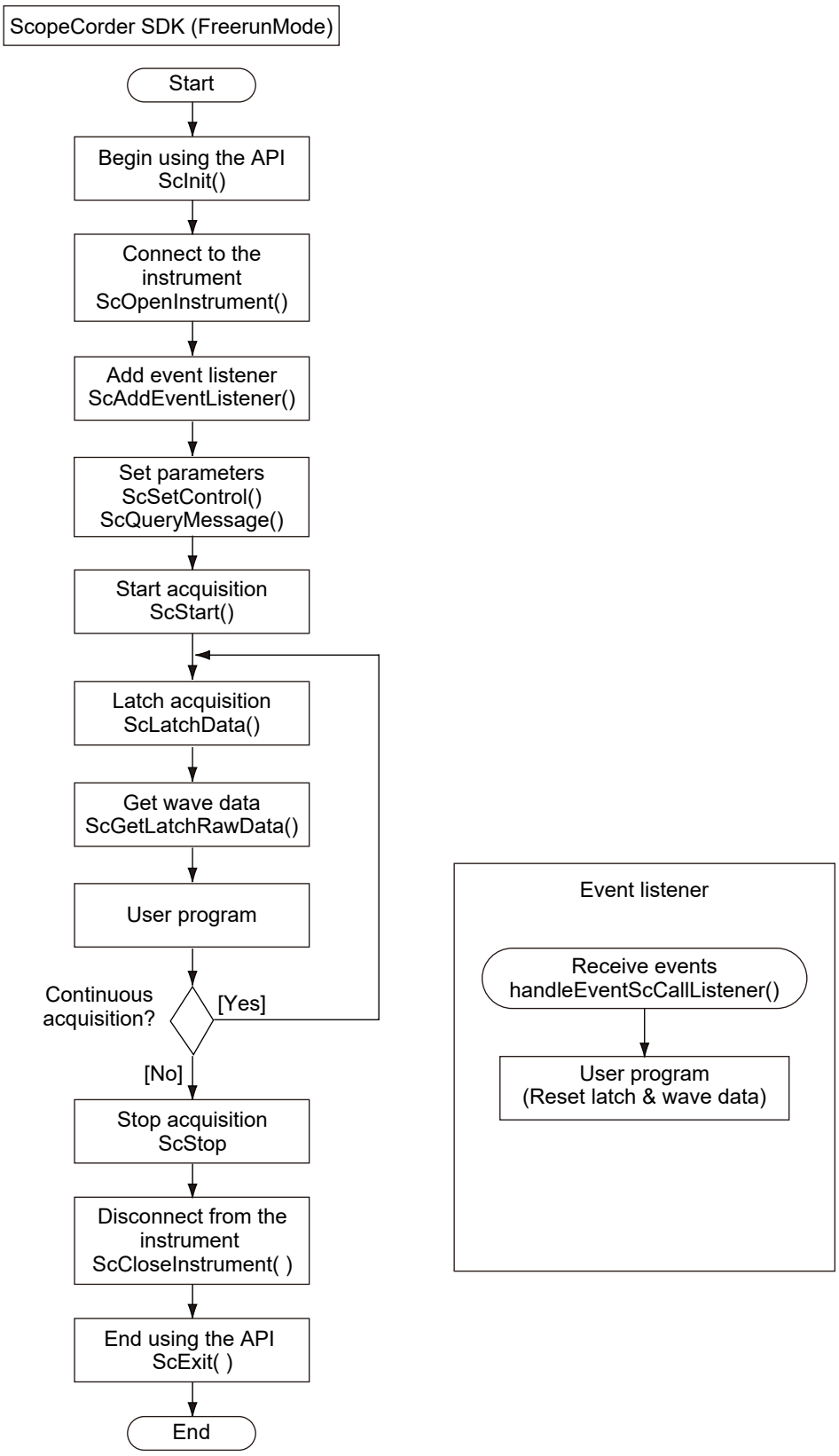
Set and query

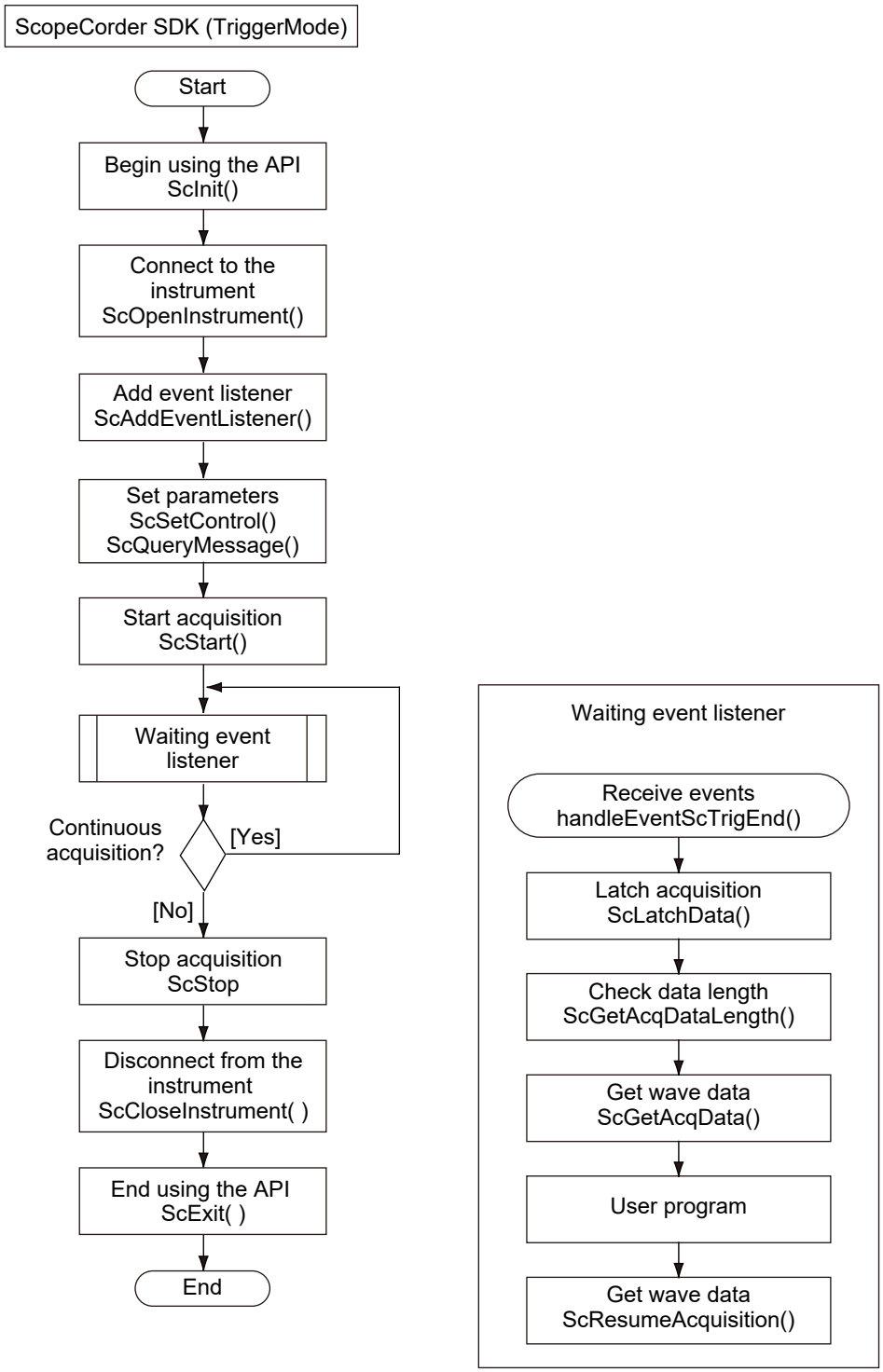
The API enables data access and file operation and transfer for data acquisition and flash acquisition by means of handles. However, when using other functions, DL950/SL2000 communication commands are sent and received using APIs such as ScSetControl and ScGetControl. For details, see section 5.3. In addition, this API is set at the start of the connection so that all responses to commands are only data without headers.

Event messages

The API has a function to notify the user with an event message that a trigger has been detected by the instrument or that a measurement has been completed, in order to improve the efficiency of application program execution. Here, this is called an event. Events are generated using service request (SRQ) interrupts from the equipment. This allows you to write programs that perform processing in response to notified events.

Data Acquisition Function




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()

Unmanaged application (free run mode)

The basic flow of using the API and a sample code for C++ (unmanaged application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).

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

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.

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

3. Add an event listener.
   In free mode, when an interface other than 10GEther is in use, data overrun can be detected. To detect overruns, use overrun events. To use overrun events, create a class that inherits the ScEventListener class, and add it to the API. Overwriting the handleEventScCallListener method causes the same method to be called when an overrun occurs. When an overrun is detected in free run mode, the data retrieved using waveform data acquisition becomes invalid (received data is no longer guaranteed). If this occurs, latch commands can be sent consecutively to clear this state.
   Note that if waveform acquisition sampling is slow and the communication environment allows data to be retrieved continuously, waveform acquisition is possible without adding overrun detection.

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

4. Start waveform acquisition.

```
ScStart(handle);
```

5. Latch (required to acquire waveforms).
   This marks the acquisition position of the waveform data.

```
ScLatchData(handle);
```

6. Get the waveform.

```
char buff[100000];
ScGetLatchRawData(handle, buff, sizeof(buff), &recieveLen);
. . .
```

   Repeat steps 5 (latch) and 6 (waveform data acquisition) during waveform acquisition.

7. Stops waveform acquisition

```
ScStop(handle);
```

8. Disconnect from the instrument (required).
   The handle is invalidated when this API command is called.

```
ScCloseInstrument(handle);
```

9. Close the API (required).

```
ScExit();
```

Managed application (free run mode)

The basic flow using the API and a sample code for C# (managed application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).
   Add ScSDKNet.dll to References of the Visual Studio Solution Explorer in advance. The name space is ScSDKNet, and the API is defined as methods in the ScSDK class.

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

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.

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

3. Add an event callback method.
   In free mode, when an interface other than 10GEther is in use, data overrun can be detected. To detect overruns, use overrun events. To use overrun events, add a callback method to the API. The same method will be called when overrun events occur. When an overrun is detected in free run mode, the data retrieved using waveform data acquisition becomes invalid (received data is no longer guaranteed). If this occurs, latch commands can be sent consecutively to clear this state.
   Note that if waveform acquisition sampling is slow and the communication environment allows data to be retrieved continuously, waveform acquisition is possible without adding overrun detection.

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

4. Start waveform acquisition.

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

5. Latch (required to acquire waveforms).
   This marks the acquisition position of the waveform data.

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

6. Get the waveform.

```
byte[] buff = new byte[100000];
int recieveLen;
api.ScGetLatchRawData<byte>(handle, buff, buff.Length,
 out receiveLen);
```

   Repeat steps 5 (latch) and 6 (waveform data acquisition) during waveform acquisition.

7. Stops waveform acquisition

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

8. Disconnect from the instrument (required).
   The handle is invalidated when this API command is called.

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

9. Close the API (required).

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

Unmanaged application (trigger mode)

The basic flow of using the API and a sample code for C++ (unmanaged application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).

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

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.

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

3. Add an event listener.
   In synchronous trigger mode, use trigger end events. To use trigger end events, create a class that inherits the ScEventListener class, and add it to the API. Overwriting the handleEventScTrigEnd method causes the same method to be called when a trigger end event occurs.
   In asynchronous trigger mode, there is no need to create or register an event listener because events do not occur.

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

4. Start waveform acquisition.

```
ScStart(handle);
```

5. Latch (required to acquire waveforms).
   Measurement information (history information) is marked.

```
ScLatchData(handle);
```

6. Get the history information.
   Read the latched acquisition count, and check whether the history has been updated. If so, set the acquisition count for reading the data.

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

7. Get the waveform.
   When waveforms are acquired in trigger mode, the number of points that can be obtained with ScGetAcqDataLength is the specified record length. The number of points depends on the record length and T/Div (sample rate). Since the size passed to ScGetAcqData is the number of bytes, determine the data point size with ScGetChannelBits in advance.

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

   Note that the maximum size that can be obtained with one ScGetAcqData is 9999999999 bytes due to communication specification limitations. Therefore, if the record length is more than 500 Mpoints (for analog modules such as voltage), data must be obtained using ScGetAcqData multiple times.

   In synchronous mode, resume acquisition (ScResumeAcquisition) when the data is acquired from all necessary channels.

   Repeat steps 5 (latch) to 7 (waveform data acquisition) during waveform acquisition.

8. Stops waveform acquisition

```
ScStop(handle);
```

9. Disconnect from the instrument (required).
   The handle is invalidated when this API function is called.

```
ScCloseInstrument(handle);
```

10. Close the API (required).

```
ScExit();
```

Managed application (trigger mode)

The basic flow using the API and a sample code for C# (managed application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).
   Add ScSDKNet.dll to References of the Visual Studio Solution Explorer in advance.
   The name space is ScSDKNet, and the API is defined as methods in the ScSDK class.

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

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.

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

3. Add an event callback method.
   In synchronous trigger mode, use trigger end events. To use trigger end events, add a callback method to the API. The same method will be called when trigger end events occur.
   In asynchronous trigger mode, there is no need to create or register an event listener because events do not occur.

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

4. Start waveform acquisition.

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

5. Latch (required to acquire waveforms).
   Measurement information (history information) is marked.

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

6. Check the history information.
   Read the latched acquisition count, and check whether the history has been updated. If so, set the acquisition count for reading the data.

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

7. Get the waveform.
   When waveforms are acquired in trigger mode, the number of points that can be obtained with ScGetAcqDataLength is the specified record length. The number of points depends on the record length and T/Div (sample rate). Since the size passed to ScGetAcqData is the number of bytes, determine the data point size with ScGetChannelBits in advance.

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

   Note that the maximum size that can be obtained with one ScGetAcqData is 9999999999 bytes due to communication specification limitations. Therefore, if the record length is more than 500 Mpoints (for analog modules such as voltage), data must be obtained using ScGetAcqData multiple times.

   In synchronous mode, resume acquisition (ScResumeAcquisition) when the data is acquired from all necessary channels.

   Repeat steps 5 (latch) to 7 (waveform data acquisition) during waveform acquisition.

8. Stops waveform acquisition

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

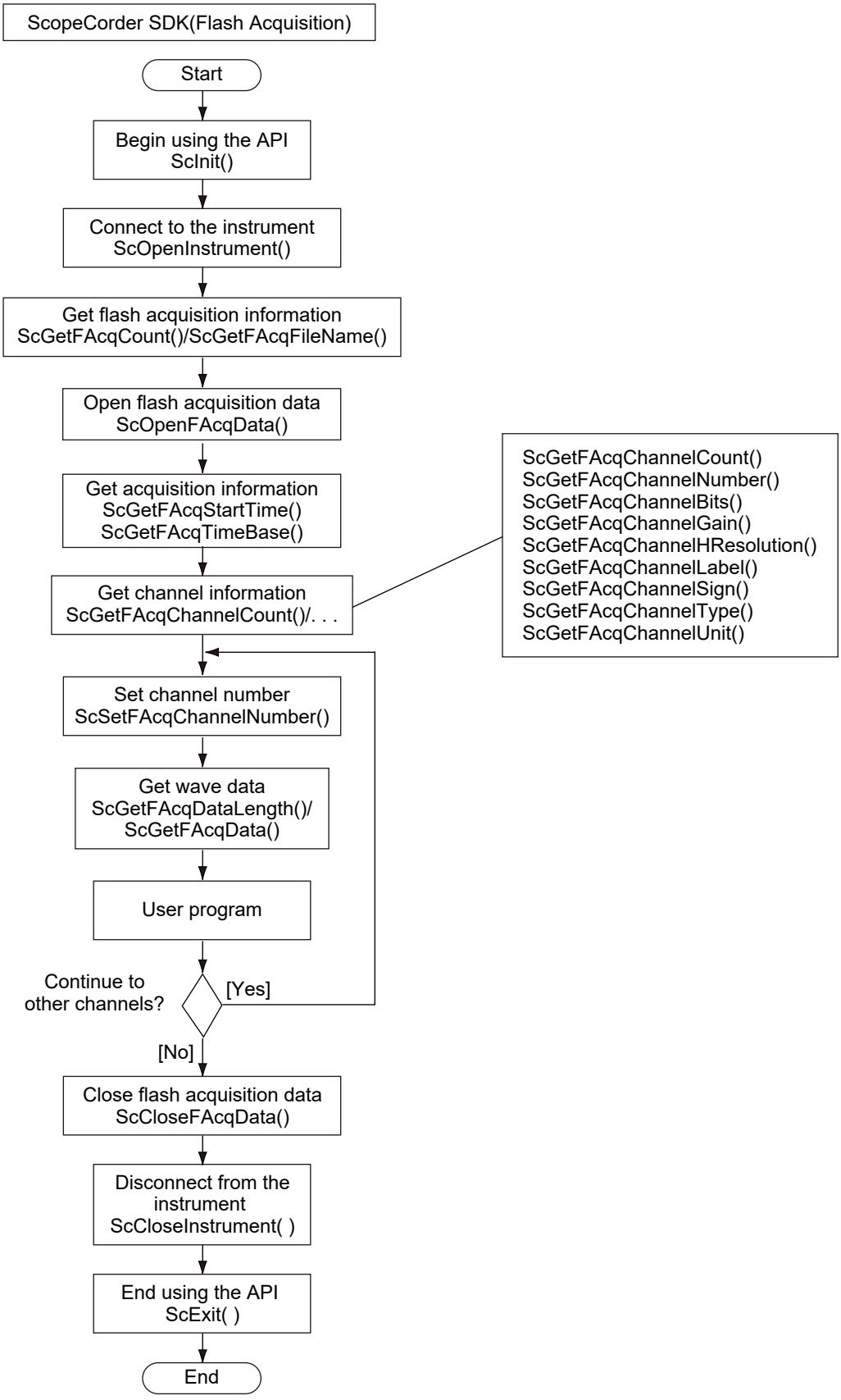
9. Disconnect from the instrument (required).
   The handle is invalidated when this API function is called.

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

10. Close the API (required).

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

Flash acquisition data access library



Unmanaged Application

The basic flow of using the API and a sample code for C++ (unmanaged application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).
```
#include "ScSDK.h"
. . .
ScInit();
. . .
```

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.
```
ScHandle handle;
ScOpenInstrument(SC_WIRE_USB, "91K225903", &handle);
```

3. Get a list of waveform data files (ScGetFAcqFileName is necessary for opening files).
   Get a list of waveform data files recorded using flash acquisition.
```
int facqCount;
char name[500][64];
ScGetFAcqCount(handle, &facqCount);
for(int i = 0; i < facqCount; i++){
  ScGetFAcqFileName(handle, i+1, &name[i][0]);
}
```

4. Open a waveform data file (required).
   Open a waveform data file you want to transmit data from. Use a file name obtained in step 3.
```
ScOpenFAcqData(handle, name);
```

5. Get the number of channels and the channel numbers in the waveform data file (channel numbers obtained with ScGetFAcqChannleNumber are necessary to get waveform data).
   Get the number of channels and the channel numbers contained in the opened waveform data file.
```
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. Get the setup information of all the channels contained in the waveform data file.
   Using the channel numbers obtained in step 5, get the setup information of all the contained channels. The obtained setup information is used for purposes such as converting the waveform data of each channel into physical values.

```
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. Get the waveform data of all the channels
   Using the channel numbers obtained in step 5, get the waveform data of all the contained channels.

```
__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;
  }
  . . .
}
```

   As shown in the example above, all the data of the contained channels can be obtained by repeating ScGetFAcqData until rcv_len becomes zero. Note that you can specify the maximum data size to read at one time using ScSetFAcqDataSize.

8. Disconnect from the instrument (required).
   The handle is invalidated when this API function is called.

```
ScCloseInstrument(handle);
```

9. Close the API (required).

```
ScExit();
```

Managed Application

The basic flow using the API and a sample code for C# (managed application) are provided below.
Error procedures are omitted.

1. Initialize the API (required).
   Add ScSDKNet.dll to References of the Visual Studio Solution Explorer in advance. The name space is ScSDKNet, and the API is defined as methods in the ScSDK class.

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

2. Open the instrument (DL950/SL2000) and create a handle (required).
   After opening the instrument, use this handle to access the instrument.

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

3. Get a list of waveform data files (ScGetFAcqFileName is necessary for opening files).
   Get a list of waveform data files recorded using flash acquisition.

```
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. Open a waveform data file (required).
   Open a waveform data file you want to transmit data from. Use a file name obtained in step 3.

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

5. Get the number of channels and the channel numbers in the waveform data file (channel numbers obtained with ScGetFAcqChannleNumber are necessary to get waveform data).
   Get the number of channels and the channel numbers contained in the opened waveform data file.

```
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. Get the setup information of all the channels contained in the waveform data file.
   Using the channel numbers obtained in step 5, get the setup information of all the contained channels. The obtained setup information is used for purposes such as converting the waveform data of each channel into physical values.

```
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. Get the waveform data of all the channels
   Using the channel numbers obtained in step 5, get the waveform data of all the contained channels.

```
__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;
  }
   . . .
}
```

   As shown in the example above, all the data of the contained channels can be obtained by repeating ScGetFAcqData until rcv_len becomes zero. Note that you can specify the maximum data size to read at one time using ScSetFAcqDataSize.

8. Disconnect from the instrument (required).
   The handle is invalidated when this API function is called.

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

9. Close the API (required).

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

3.1 Definition of Class

This section explains the API class definitions.

Class ScEventListener

Function:

Event listener class for receiving events (C++ only)

Syntax:

```
class ScEventListener {
public:
    /*!
    * \brief Overrun handler
    * \param handle API handle

    * \param\ reserve
    */

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

Details:

The events that you can register are the over run events for free run mode and the trigger end events for synchronous trigger mode.
Overwriting handleEventScCallListener causes the same method to be called automatically when an overrun event occurs.
Overwriting handleEventScTrigEnd causes the same method to be called automatically when a trigger end event occurs.
Use ScAddEventListener to create instances.

3.2 Definition of Constants

SC_SUCCESS

Description:

Normal

Syntax:

#define SC_SUCCESS

Details:

Definition of a result returned by API functions

SC_ERROR

Description:

Errors

Syntax:

#define SC_ERROR

Details:

Definition of a result returned by API functions

SC_UNOPENED

Description:

File not specified

Syntax:

#define SC_UNOPENED

Details:

Definition of a result returned by API functions (flash acquisition data access library function)

SC_USE_ACQMEMORY

Description:

Measurement data exists in ACQ memory

Syntax:

#define SC_USE_ACQMEMORY

Details:

Definition of a result returned by ScOpenFAcqData

SC_ERR_UNOPENED

Description:

Error when file not specified

Syntax:

#define SC_ERR_UNOPENED

Details:

Definition of a result returned by API functions (flash acquisition data access library function)

SC_ERR_RUNNING

Description:

Error when the instrument to which the connection was made is measuring

Syntax:

#define SC_ERR_RUNNING

Details:

Definition of the return value when Open functions such as ScOpenInstrument are executed

SC_ERR_SYNC_CONN

Description:

Error when the instrument to which the connection was made is in multi-unit synchronization connection

Syntax:

#define SC_ERR_SYNC_CONN

Details:

Definition of the return value when Open functions such as ScOpenInstrument are executed

SC_ERR_SYNC_SUB

Description:

Error for a sub unit in multi-unit synchronization

Syntax:

#define SC_ERR_SYNC_SUB

Details:

Definition of the return value when Open functions such as ScOpenInstrument and data acquisition related functions are executed.

SC_ERR_RECORDER

Description:

Error when the instrument to which the connection was made is in recorder mode

Syntax:

#define SC_ERR_RECORDER

Details:

Definition of the return value when Open functions such as ScOpenInstrument are executed

SC_ERR_MODE

Description:

Error when the mode and instrument settings differ from the those specified when reconnecting

Syntax:

#define SC_ERR_MODE

Details:

Definition of the return value when ScReopenInstrument is executed

SC_ERR_NOTAPPLICABLE

Description:

Error when the connected instrument is other than the DL950/SL2000 series or an incompatible version.

Syntax:

#define SC_ERR_NOTAPPLICABLE

Details:

Definition of the return value when Open functions such as ScOpenInstrument are executed

SC_ERR_NODATA

Description:

Error when there is no file name for the waveform data corresponding to the specified flash acquisition number or there is no waveform to save

Syntax:

#define SC_ERR_NODATA

Details:

Definition of the return value when ScGetFAcqFileName and ScSaveTriggerWDF are executed

SC_ERR_PARAMETER

Description:

Error when the function parameter is invalid

Syntax:

#define SC_ERR_PARAMETER

Details:

Definition of the value returned when a function parameter is invalid.

SC_WIRE_USBTMC

Description:

USB wire type (YTUSB)

Syntax:

#define SC_WIRE_USBTMC

Details:

Definition of a wire type for connecting to the DL950/SL2000 series
* Select this to use a USB (TMCTL standard driver) connection.

SC_WIRE_VISAUSB

Description:

USB wire type (VISAUSB)

Syntax:

#define SC_WIRE_VISAUSB

Details:

Definition of a wire type for connecting to the DL950/SL2000 series
* Select this to use a USB (when a VISA driver is in use) connection.

SC_WIRE_VXI11

Description:

Ethernet wire type (VXI11)

Syntax:

#define SC_WIRE_VXI11

Details:

Definition of a wire type for connecting to the DL950/SL2000 series
* Select this to use GigaBitEther.

SC_WIRE_HISLIP

Description:

Ethernet wire type (HiSLIP)

Syntax:

#define SC_WIRE_HISLIP

Details:

Definition of a wire type for connecting to the DL950/SL2000 series
* Select this to use the 10G high-speed data transmission mode.

SC_FREERUN

Description:

Free run operation

Syntax:

#define SC_FREERUN

Details:

Specify this to implement waveform acquisition in free run mode.
Data received from the DL950/SL2000 is passed as-is to the program as block data.

SC_TRIGGER

Description:

Synchronous trigger mode

Syntax:

#define SC_TRIGGER

Details:

Specify this to acquire waveform data in synchronous trigger mode.
The DL950/SL2000 waveform acquisition sequence is explicitly controlled using the API.

SC_TRIGGER_ASYNC

Description:

Asynchronous trigger mode

Syntax:

#define SC_TRIGGER_ASYNC

Details:

Specify this to acquire waveform data in asynchronous trigger mode.
Waveform acquisition will take place on the DL950/SL2000 regardless of whether data acquisition has been completed.

SC_NOMODE

Description:

No mode specified for connection

Syntax:

#define SC_NOMODE

Details:

Specify this when connecting to the instrument under measurement or using the Flash Acquisition Data Access Library function.

SC_EVENTTYPE_OVERRUN

Description:

Event type (overrun)

Syntax:

#define SC_EVENTTYPE_OVERRUN

Details:

Specify the event type for registering an overrun event callback in free run mode.
This is used only with the .NET version (C#).

SC_EVENTTYPE_TRIGGEREND

Description:

Event type (trigger end)

Syntax:

#define SC_EVENTTYPE_TRIGEREND

Details:

Specify the event type for registering a trigger end event callback in trigger mode.
This is used only with the .NET version (C#).

SC_SIZE_16MB

Description:

Data transmission size at 16 MiB

Syntax:

#define SC_SIZE_16MB

Details:

Definition of the data transmission size

SC_SIZE_32MB

Description:

Data transmission size at 32 MiB

Syntax:

#define SC_SIZE_32MB

Details:

Definition of the data transmission size

SC_SIZE_64MB

Description:

Data transmission size at 64 MiB

Syntax:

#define SC_SIZE_64MB

Details:

Definition of the data transmission size

SC_SIZE_128MB

Description:

Data transmission size at 128 MiB

Syntax:

#define SC_SIZE_128MB

Details:

Definition of the data transmission size

SC_SIZE_256MB

Description:

Data transmission size at 256 MiB

Syntax:

#define SC_SIZE_256MB

Details:

Definition of the data transmission size

SC_SIZE_512MB

Description:

Data transmission size at 512 MiB

Syntax:

#define SC_SIZE_512MB

Details:

Definition of the data transmission size

SC_10GMODE_ON

Description:

10Gbps high-speed transmission mode enabled

Syntax:

#define SC_10GMODE_ON

Details:

Used to set the 10Gbps high-speed transmission mode.

SC_10GMODE_OFF

Description:

10Gbps high-speed transmission mode disabled

Syntax:

#define SC_10GMODE_OFF

Details:

Used to set the 10Gbps high-speed transmission mode.

SC_DRIVE_IDRIVE

Description:

Set the current drive to IDRive.

Syntax:

#define SC_DRIVE_IDRIVE

Details:

Used to set the current drive.

SC_DRIVE_NETWORK

Description:

Set the current drive to NETWork.

Syntax:

#define SC_DRIVE_NETWORK

Details:

Used to set the current drive.

SC_DRIVE_SD

Description:

Set the current drive to SD.

Syntax:

#define SC_DRIVE_SD

Details:

Used to set the current drive.

SC_DRIVE_USB_0

Description:

Set the current drive to USB-0.

Syntax:

#define SC_DRIVE_USB_0

Details:

Used to set the current drive.

SC_DRIVE_USB_1

Description:

Set the current drive to USB-1.

Syntax:

#define SC_DRIVE_USB_1

Details:

Used to set the current drive.

SC_DRIVE_FLASH

Description:

Set the current drive to FLASh.

Syntax:

#define SC_DRIVE_FLASH

Details:

Used to set the current drive.

SC_FILE_ETE_ALL

Description:

Set the extension to all for retrieving files.

Syntax:

#define SC_FILE_ETE_ALL

Details:

Used when getting a list of files.

SC_FILE_ETE_SET

Description:

Specify the *.SET extension for retrieving files.

Syntax:

#define SC_FILE_ETE_SET

Details:

Used when getting a list of files.

SC_FILE_ETE_WDF

Description:

Specify the *.WDF extension for retrieving files.

Syntax:

#define SC_FILE_ETE_WDF

Details:

Used when getting a list of files.

SC_FILE_ETE_BMP

Description:

Specify the *.BMP extension for retrieving files.

Syntax:

#define SC_FILE_ETE_BMP

Details:

Used when getting a list of files.

SC_FILE_ETE_PNG

Description:

Specify the *.PNG extension for retrieving files.

Syntax:

#define SC_FILE_ETE_PNG

Details:

Used when getting a list of files.

SC_FILE_ETE_JPG

Description:

Specify the *.JPG extension for retrieving files.

Syntax:

#define SC_FILE_ETE_JPG

Details:

Used when getting a list of files.

SC_FILE_ETE_SNP

Description:

Specify the *.SNP extension for retrieving files.

Syntax:

#define SC_FILE_ETE_SNP

Details:

Used when getting a list of files.

SC_FILE_ETE_SBL

Description:

Specify the *.SBL extension for retrieving files.

Syntax:

#define SC_FILE_ETE_SBL

Details:

Used when getting a list of files.

SC_FILE_ETE_CSV

Description:

Specify the *.CSV extension for retrieving files.

Syntax:

#define SC_FILE_ETE_CSV

Details:

Used when getting a list of files.

SC_FILE_ETE_MAT

Description:

Specify the *.MAT extension for retrieving files.

Syntax:

#define SC_FILE_ETE_MAT

Details:

Used when getting a list of files.

SC_SYNC_OFF

Description:

Indicates that the instrument is not using the multi-unit synchronization feature.

Syntax:

#define SC_SYNC_OFF

Details:

Used when getting the handle list, device list, or status information.

SC_SYNC_CONN

Description:

Indicates that the instrument is in multi-unit synchronization connection standby.

Syntax:

#define SC_SYNC_CONN

Details:

Used when getting the device list or status information.

SC_SYNC_MAIN

Description:

Indicates that the instrument is running as the main unit.

Syntax:

#define SC_SYNC_MAIN

Details:

Used when getting the handle list, device list, or status information.

SC_SYNC_SUB

Description:

Indicates that the instrument is running as a sub unit.

Syntax:

#define SC_SYNC_SUB

Details:

Used when getting the handle list, device list, or status information.

SC_STAT_STOPPED

Description:

Indicates that measurement is stopped on the instrument.

Syntax:

#define SC_STAT_STOPPED

Details:

Used when getting the status information.

SC_STAT_RUNNING

Description:

Indicates that the instrument is measuring.

Syntax:

#define SC_STAT_RUNNING

Details:

Used when getting the status information.

SC_STAT_INTERNAL

Description:

Indicates that the instrument's time base is set to internal.

Syntax:

#define SC_ STAT_INTERNAL

Details:

Used when getting the status information.

SC_STAT_EXTERNAL

Description:

Indicates that the instrument's time base is set to external.

Syntax:

#define SC_STAT_EXTERNAL

Details:

Used when getting the status information.

SC_STAT_SSD

Description:

Indicates that the instrument's real-time recording is set to SSD recording.

Syntax:

#define SC_STAT_SSD

Details:

Used when getting the status information.

SC_STAT_FACQ

Description:

Indicates that the instrument's real-time recording destination is flash acquisition.

Syntax:

#define SC_STAT_FACQ

Details:

Used when getting the status information.

SC_STAT_TRIGGER

Description:

Indicates that the instrument's acquisition mode is trigger.

Syntax:

#define SC_STAT_TRIGGER

Details:

Used when getting the status information.

SC_STAT_FREERUN

Description:

Indicates that the instrument's acquisition mode is free run.

Syntax:

#define SC_STAT_FREERUN

Details:

Used when getting the status information.

SC_STAT_OFF

Description:

Indicates that the instrument's status setting is off.

Syntax:

#define SC_STAT_OFF

Details:

Used when getting the status information.

SC_STAT_ON

Description:

Indicates that the instrument's status setting is on.

Syntax:

#define SC_STAT_ON

Details:

Used when getting the status information.

SC_STAT_ST1

Description:

Indicates that the instrument's internal storage option is ST1.

Syntax:

#define SC_STAT_ST1

Details:

Used when getting the status information.

SC_STAT_ST2

Description:

Indicates that the instrument's internal storage option is ST2.

Syntax:

#define SC_STAT_ST2

Details:

Used when getting the status information.

SC_SORT_NAME_ASC

Description:

Get a list of files in ascending order by file name.

Syntax:

#define SC_SORT_NAME_ASC

Details:

Used when getting a list of files.

SC_SORT_NAME_DESC

Description:

Get a list of files in descending order by file name.

Syntax:

#define SC_SORT_NAME_DESC

Details:

Used when getting a list of files.

SC_SORT_DATE_ASC

Description:

Get a list of files in ascending order file update date.

Syntax:

#define SC_SORT_DATE_ASC

Details:

Used when getting a list of files.

SC_SORT_DATE_DESC

Description:

Get a list of files in descending order by file update date.

Syntax:

#define SC_SORT_DATE_DESC

Details:

Used when getting a list of files.

SC_SORT_SIZE_ASC

Description:

Get a list of files in ascending order by file size.

Syntax:

#define SC_SORT_SIZE_ASC

Details:

Used when getting a list of files.

SC_SORT_SIZE_DESC

Description:

Get a list of files in descending order by file size.

Syntax:

#define SC_SORT_SIZE_DESC

Details:

Used when getting a list of files.

3.3 Definitions of Data Structures

HandleList

Description:

Handle list

Syntax:

```
struct HandleList {
    ScHandle  handle;     // handle of the connected instrument
    int       sync;       // multi-unit synchronization state of the connected instrument
    char      unit[32];   // unit name
};
```

Details:

Used when getting a list of handles.
The following fixed values are stored for each parameter.

Sync	SC_SYNC_OFF / SC_SYNC_MAIN / SC_SYNC_SUB

DeviceList

Description:

Device list

Syntax:

```
struct DeviceList {
    char      adr[64];    // address
    int       sync;       // multi-unit synchronization state of the connected instrument
    char      name[32];   // instrument name
    char      unit[32];   // unit name
};
```

Details:

Used when getting a list of devices.
The following fixed values are stored for each parameter.

Sync	SC_SYNC_OFF / SC_SYNC_CONN / SC_SYNC_MAIN / SC_SYNC_SUB

StatusInfo

Description:

Status information

Syntax:

```
struct StatusInfo {
    int       Running;          // measurement state
    int       Sync;             // multi-unit synchronization connection setting
    int       TimeBase;         // time base
    int       RealTime;         // real-time recording setting
    int       AcqMode;          // acquisition mode
    int       DualCapture;      // dual capture setting
    int       GONogo;           // GONogo setting
    int       StorageOpt;       // storage option
};
```

Details:

Used when getting the device settings.
The following fixed values are stored for each parameter.

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

Description:

File information

Syntax:

```
struct FileInfo {
    char          name[256];     // file name
    unsigned int  size;          // file size
    char          date[10];      // file storage date
    char          time[5];       // file storage time
    char          rw[5];         // read/write
};
```

Details:

Used when getting the file list and file list information.

4.1 Common API

ScInit

Description:

Initialize the API

Syntax:

[C++]

ScResult ScInit(void);

[C#]

int ScInit();

Parameters:

No

Return value:

SC_SUCCESS	Success
SC_ERROR	Initialization error (already initialized)

Details:

Call once at the start of using the library.

Example [C++]:

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

Example [C#]:

```
using ScSDKNet;
...
ScSDKNet.ScSDK api = new ScSDKNet.ScSDK();
if (api.ScInit() == ScSDK.SC_SUCCESS)
{
  ...
}
```

ScExit

Description:

End using the API

Syntax:

[C++]

ScResult ScExit(void);

[C#]

int ScExit();

Parameters:

No

Return value:

SC_SUCCESS	Success
SC_ERROR	Error (already terminated or not initialized)

Details:

Call once at the end of using the API.

ScOpenInstrument

Description:

Open the instrument

Syntax:

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

Parameters:

[IN] wire	Wire type	
	SC_WIRE_USBTMC	USBTMC (YTUSB)
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	Connection destination address (instrument serial number for USB)	
[IN] mode	Connection mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode
	SC_NOMDOE	No connection mode
[OUT] rHndl	Instrument handle	

Return value:

SC_SUCCESS	Connection successful
SC_ERROR	Connection error
SC_ERR_RUNNING	Error during measurement
SC_ERR_SYNC_CONN	Error during multi-unit synchronization connection
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error
SC_ERR_RECORDER	Recorder mode error
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Connects to the instrument and returns the instrument handle.
This handle is passed to the APIs to communicate with the instrument.
When a connection is established, the waveform acquisition conditions of the measuring instrument are set automatically according to the mode parameter.
When SC_NOMODE is selected, the currently set connection mode is maintained.
If a mode other than SC_NOMODE is selected, an error is generated if the instrument is measuring.

Note:

Multiple connections to a single instrument is not possible.
To use 10Gbps Ethernet, select SC_WIRE_HISLIP.
Connection is not possible when to DL950/SL2000 is in recorder mode.

Example [C++]:

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

Example [C#]:

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

ScReopenInstrument

Description:

Open the instrument (possible on an instrument that is measuring)

Syntax:

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

Parameters:

[IN] wire	Wire type	
	SC_WIRE_USBTMC	USBTMC (YTUSB)
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	Connection destination address (instrument serial number for USB)	
[IN] mode	Connection mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode
[OUT] rHndl	Instrument handle	

Return value:

SC_SUCCESS	Connection successful
SC_ERROR	Connection error
SC_ERR_SYNC_CONN	Error during multi-unit synchronization connection
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error
SC_ERR_RECORDER	Recorder mode error
SC_ERR_MODE	Wrong mode error
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Connects to the instrument that is currently measuring and returns the instrument handle. This handle is passed to the APIs to communicate with the instrument.
When measurement is stopped, the specified mode parameter is automatically passed to the measuring instrument.

Note:

Multiple connections to a single instrument is not possible.
To use 10Gbps Ethernet, select SC_WIRE_HISLIP.
Connection is not possible when to DL950/SL2000 is in recorder mode.
If a measurement is in progress and the specified mode parameter is different from the state of the measuring instrument, an error is returned.

Example [C++]:

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

Example [C#]:

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

ScCloseInstrument

Description:

Close the instrument

Syntax:

[C++]

ScResult ScCloseInstrument(ScHandle hndl);

[C#]

int ScCloseInstrument(int hndl);

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Error (not connected or already disconnected)

Details:

Disconnects from the instrument connected using ScOpenInstrument.
If the instrument is measuring, disconnection takes time, but it can be done in a short time by decreasing the timeout period with ScSetTriggerTimeout.

Note:

The handle is invalidated when this API command is called.

ScOpenInstrumentEx

Description:

Connect to an instrument in multi-unit synchronization

Syntax:

[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);

Parameters:

[IN] wire	Wire type	
	SC_WIRE_USBTMC	USBTMC (YTUSB)
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	Connection destination address of the main unit (instrument serial number in the case of USB)	
[IN] mode	Connection mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode
	SC_NOMDOE	No connection mode
[OUT] List	Instrument handle list	
[IN] max	Maximum size of the instrument handle list	
[OUT] ListCount	Number of instrument handle lists	

Return value:

SC_SUCCESS	Connection successful
SC_ERROR	Connection error
SC_ERR_RUNNING	Error during measurement
SC_ERR_SYNC_CONN	Error during multi-unit synchronization connection
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error
SC_ERR_RECORDER	Recorder mode error
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Connects to the sub units from the address of the main unit in multi-unit synchronization and obtains the handle of each instrument.
Performs the connection process of this API for the instruments using the instrument handles obtained from the API of the TMCTL library.
This handle is passed to the APIs to communicate with the instrument.
When a connection is established, the waveform acquisition conditions of the measuring instrument are set automatically according to the mode parameter.
Specify the maximum size of the array for the max parameter.

Note:

Multiple connections to a single instrument is not possible.
To use 10Gbps Ethernet, select SC_WIRE_HISLIP.
Connection is not possible when to DL950/SL2000 is in recorder mode.
The only instrument handle used in this API is assumed to be that of the DL950/SL2000 connected via the TMCTL library API. It will not work properly with an instrument handle acquired otherwise.

ScReopenInstrumentEx

Description:

Connect to an instrument in multi-unit synchronization (possible on an instrument that is measuring)

Syntax:

[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);
```

Parameters:

[IN] wire	Wire type	
	SC_WIRE_USBTMC	USBTMC (YTUSB)
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
	SC_WIRE_HISLIP	HiSLIP
[IN] address	Connection destination address (instrument serial number for USB)	
[IN] mode	Connection mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode
[OUT] List	Instrument handle list	
[IN] max	Maximum size of the instrument handle list	
[OUT] ListCount	Number of instrument handle lists	

Return value:

SC_SUCCESS	Connection successful
SC_ERROR	Connection error
SC_ERR_SYNC_CONN	Error during multi-unit synchronization connection
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error
SC_ERR_RECORDER	Recorder mode error
SC_ERR_MODE	Wrong mode error
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Connects to the instrument that is currently measuring and returns the instrument handle. This handle is passed to the APIs to communicate with the instrument.
When measurement is stopped, the specified mode parameter is automatically passed to the measuring instrument.

Note:

Multiple connections to a single instrument is not possible.
To use 10Gbps Ethernet, select SC_WIRE_HISLIP.
Connection is not possible when to DL950/SL2000 is in recorder mode.
If a measurement is in progress and the specified mode parameter is different from the state of the measuring instrument, an error is returned.

ScCloseInstrumentEx

Description:

Disconnect from an instrument in multi-unit synchronization

Syntax:

[C++]

ScResult ScCloseInstrumentEx(HandleList* List, int ListCount);

[C#]

int ScCloseInstrumentEx(HandleList[] List, int ListCount);

Parameters:

[IN] List	Instrument handle list
[IN] ListCount	Number of instrument handle lists

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Disconnects from the instrument connected using ScOpenInstrumentEx.

Note:

The handle of this API is invalidated when this API function is called.

ScSetControl

Description:

Send a communication command

Syntax:

[C++]

ScResult ScSetControl(ScHandle hndl, char* command);

[C#]

int ScSetControl(int hndl, string command);

Parameters:

[IN] hndl	Instrument handle
[IN] command	Communication command string

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Sends a communication command to the instrument.

Note:

The return value cannot be used to determine communication command errors. It only indicates whether the command was sent successfully.

ScGetControl

Description:

Receive a response to a communication command

Syntax:

[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);

Parameters:

[IN] hndl	Instrument handle
[OUT] buff	Receive buffer
[IN] buffLen	Buffer size
[OUT] receiveLen	Length of the received response

Return value:

SC_SUCCESS	Success
SC_ERROR	Error (no data to be received)

Details:

Receives a response to a communication command sent in advance from the instrument.

Note:

An error occurs if a communication command has not been sent in advance.

Example [C++]:

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

Example [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

Description:

Receive binary data

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle	
[IN] command	Communication command for requesting binary data Specify 0 (null pointer) to receive data being received.	
[OUT] buff	Buffer for receiving binary data	
[IN] buffLen	Size of the buffer for receiving binary data (bytes)	
[OUT] receiveLen	Size of the received binary data (bytes)	
[OUT] endFlg	Receive end flag	
	0	Receiving (remaining data available)
	1	Receive end

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Sends a command for querying binary data and receives the response.
When the buffer size specified by buffLen is smaller than the size of the binary data actually received, endFlg is set to zero.
To continue receiving binary data when the ScGetBinaryData, ScGetLatchAcqData, ScGetAcqData's receive complete flag is not 1, set the command parameter to 0 (null pointer).

Note:

The behavior when a command that does not send binary data is specified is undefined.

Example [C++]:

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

Example [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

Description:

Issue a command and receive its response

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle
[IN] command	Communication Commands
[OUT] buff	Receive buffer
[IN] buffLen	Length of the receive buffer (in bytes) Length to retrieve in the case of .NET version
[OUT] receiveLen	Length of the received response

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

You can perform communication command transmission and response reception with this single API method.

Note:

You cannot use this API method for commands that do not return responses.
In the case of C# (.NET version), specify the number of bytes to receive, not the length of the buffer size.

Example [C#]:

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

Example [C#]:

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

ScSet10GMode

Description:

Set the 10Gbps high-speed data transmission mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle	
[IN] onoff	10Gbps high-speed data transmission mode setting	
	0	10Gbps high-speed data transmission mode disabled
	1	10Gbps high-speed data transmission mode enabled

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Sets whether to use hardware-driven 10Gbps high-speed data transmission for data acquisition or flash acquisition data transmission. This command can be used when the 10G option is installed.

Note:

This command is available when a 10Gbps Ethernet connection is established and the wire type is set to HiSLIP.
Execute this command before starting waveform acquisition. (ScStart). You cannot change this during waveform acquisition.
Execute this command before executing ScGetFAcqData. If you change the setting while transmission is in progress, normal transmission may be impeded. In such case, close the instrument, and start over.
Data can be transferred via 10Gbps Ethernet even if this mode is disabled, but if used in data acquisition, overruns are more likely to occur due to reduced transmission performance.
This function is disabled for file transfers; software transfer is used instead. Even with a 10 Gbps Ethernet connection, the data rate may be slower than with a 1 Gbps Ethernet connection.

ScGet10GMode

Description:

Get the 10Gbps high-speed data transmission mode setting.

Syntax:

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

Parameters:

[IN] hndl	Instrument handle	
[OUT] onoff	10Gps high-speed data transmission mode setting	
	0	10Gps high-speed data transmission mode disabled
	1	10Gps high-speed data transmission mode enabled

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Checks whether hardware-driven 10Gbps high-speed data transmission mode is enabled for data transmission.

ScSearchDevices

Description:

Get a list of instruments connected to the specified line

Syntax:

[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);

Parameters:

[IN] wire	Wire type	
	SC_WIRE_USBTMC	USBTMC (YTUSB)
	SC_WIRE_VISAUSB	VISAUSB
	SC_WIRE_VXI11	VXI-11
[OUT] list	List of instruments	
[IN] max	Maximum size of the instrument list	
[OUT] deviceCount	Number of instrument lists	
[IN] option	NETWork option string	

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_NOTAPPLICABLE	Target line type error
SC_ERR_PARAMETER	Error in the maximum size of the instrument list

Details:

The maximum number of instrument lists that can be retrieved is 128.
Specify the maximum size of the array for the max parameter.
This can be used with DL950/SL2000 with firmware version 2.01 or later.

ScGetStatusInfo

Description:

Get the status information of the connected destination

Syntax:

[C++]

ScResult ScGetStatusInfo(ScHandle hndl, StatusInfo *statusInfo);

[C#]

int ScGetStatusInfo(int hndl, out StatusInfo statusInfo);

Parameters:

[IN] hndl	Instrument handle
[IN] statusInfo	Status information of the connected destination

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the status information of connected instruments.

ScTmcSetTimeout

Description:

Set the TMCTL timeout period

Syntax:

[C++]

ScResult ScTmcSetTimeout(ScHandle hndl, int timeout);

[C#]

int ScTmcSetTimeout(ScHandle hndl, int timeout);

Parameters:

[IN] hndl	Instrument handle
[IN] timeout	TMCTL timeout value (100-ms resolution) (0 to 65536)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

The default timeout period is 30 seconds.

If set to 0, the timeout period is set to none.

4.2 Data Acquisition Function API

ScSetMeasuringMode

Description:

Setting the Measurement Mode

Syntax:

[C++]

ScResult ScSetMeasuringMode(ScHandle hndl, int mode);

[C#]

int ScSetMeasuringMode(int hndl, int mode);

Parameters:

[IN] hndl	Instrument handle	
[IN] mode	Measuring Mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_RUNNING	Error during measurement
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error

Details:

Set the measurement mode. This API is valid only when stopped.

ScSetMeasuringModeEx

Description:

Set the measurement mode for multi-unit synchronization

Syntax:

[C++]

ScResult ScSetMeasuringModeEx(HandleList* List, int num, int mode)

[C#]

int ScSetMeasuringModeEx(HandleList[] List, int num, int mode)

Parameters:

[IN] List	Instrument handle list	
[IN] num	Number of instrument handle lists	
[IN] mode	Measuring Mode	
	SC_FREERUN	Free run
	SC_TRIGGER	Synchronous trigger mode
	SC_TRIGGER_ASYNC	Asynchronous trigger mode

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_RUNNING	Error during measurement

Details:

Set the measurement mode. This API is valid only when stopped.

ScStart

Description:

Start waveform acquisition

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error

Details:

Starts waveform acquisition. (Sends a Start command.)

ScStartEx

Description:

Start waveform acquisition on instruments connected in multi-unit synchronization

Syntax:

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

Parameters:

[IN] List	Instrument handle list
[IN] num	Number of instrument handle lists

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Starts waveform acquisition. (Sends a Start command.)

ScStop

Description:

Stop waveform acquisition

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_SYNC_SUB	Multi-unit synchronization sub unit connection error

Details:

Stops waveform acquisition. (Sends a Stop command.)

ScStopEx

Description:

Stop waveform acquisition on instruments connected in multi-unit synchronization

Syntax:

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

Parameters:

[IN] List	Instrument handle list
[IN] num	Number of instrument handle lists

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Stops waveform acquisition. (Sends a Stop command.)

ScLatchData

Description:

Latch the waveform data

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Marks the present acquisition position of the waveform data in the instrument.
In free run mode, this position is used as a reference for getting waveform data.
In trigger mode, history information (acquisition information) is also marked.

ScLatchDataEx

Description:

Latch the waveform data of instruments connected in multi-unit synchronization

Syntax:

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

Parameters:

[IN] List	Instrument handle list
[IN] num	Number of instrument handle lists

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Marks the present acquisition position of the waveform data in the instrument.
In free run mode, this position is used as a reference for getting waveform data.
In trigger mode, history information (acquisition information) is also marked.

ScGetLatchRawData

Description:

Get latched waveform data in free run mode

Syntax:

[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);

Parameters:

[IN] hndl	Instrument handle
[OUT] buff	Save buffer
[IN] buffLen	Length of save buffer
[OUT] receiveLen	Size of the received binary data (bytes)
[OUT] endFlg	Receive end flag
	0 Receiving (remaining data available)
	1 Receive end

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets latched waveform data.

When the buffer size specified by buffLen is smaller than the size of the binary data actually received, endFlg is set to zero.

Note:

The waveform data contains data of all measurement channels and is provided in block format. For details on the block format, see “ScGetLatchRawData Data Structure” in section 5.1.

The returned waveform data is an AD value.

To convert to physical values, an appropriate data conversion is necessary according to the data type obtained with ScGetChannelType. The following formula is used.

Physical value = AD value × Gain + Offset

(Gain can be obtained with ScGetChannelGain and Offset with ScGetChannelOffset)

For the buffer size, see “Required memory size,” and specify a sufficient size.

10G high-speed data

If endFlag is 0, use ScGetBinaryData to receive the rest of the data.

You can use this method when SC_FREERUN mode is specified.

Example [C++]:

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

Example [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

Description:

Get the waveform data position of a specified channel from the data retrieved with ScGetLatchRawData

Syntax:

[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);

Parameters:

[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[IN] buff	Buffer containing data in block format
[IN] length	Size of the buffer containing data in block format
[OUT] chOffset	Offset position (number of bytes) to the head of the channel data
[OUT] chSize	Channel data size (number of bytes)
[OUT] timeSec	Time (UnixTime) at the head of the retrieved data
[OUT] timeTikc	Time (nanoseconds) at the head of the retrieved data

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the data position of the specified channel from the retrieved waveform data (block format).

The head time of the retrieved data is also obtained.

Programming tips:

When you use ScGetChAcqData to retrieve channel data in order to prevent data overruns when acquiring waveforms at a high sampling rate, we recommend analyzing the retrieved data using a thread different from ScGetLatchRawData.

Further, when you acquire waveforms using 10G high-speed data streaming, we recommend not using ScGetChAcqData during waveform acquisition in order to prevent data overruns but rather using ScGetLatchRawData to only retrieve data and then using ScGetChAcqData to retrieve channel data after the waveform acquisition is completed.

Note:

Prepare a buffer large enough to store the channel data. Calculate the necessary buffer size based on the data size per point using ScGetChannelBits and the interval between latches.
Since the waveform data is AD values, to convert to physical values, an appropriate data conversion is necessary according to the data type obtained with ScGetChannelType.
The following formula is used.
Physical value = AD value × Gain + Offset
(Gain can be obtained with ScGetChannelGain and Offset with ScGetChannelOffset)
If the specified channel data is not available, an error will occur.
If there is no relevant channel data between latches, the data size will be 0.
For details on the block format, see “ScGetLatchRawData Data Structure” in section 5.1.
You can use this method when SC_FREERUN mode is specified.

Example [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) {
       ...
    }
  ...
  }
```

Example [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

Description:

Get latched waveform data in trigger mode

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] buff	Save buffer
[IN] buffLen	Length of save buffer
[OUT] receiveLen	Length of the acquired data (in bytes)
[OUT] endFlg	Receive end flag
	0 Receiving (remaining data available)
	1 Receive end
[OUT] timeSec	Time (UnixTime) at the head of the retrieved data
[OUT] timeTikc	Time (nanoseconds) at the head of the retrieved data

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets latched waveform data.
The head time of the waveform data is also obtained.
For an overview of the operation when acquiring waveforms in trigger mode using this API, see section 5.1, “Trigger Mode.”
When the buffer size specified by buffLen is smaller than the size of the binary data actually received, endFlg is set to zero. In this case, use ScGetBinaryData to receive the rest of the data.
When waveforms are acquired using external sampling, timeSec and timeTick stores sample count values, not time information. For details, see section 5.1, “Trigger Mode.”

Note:

ScGetAcqDataLength need to be called immediately before calling this method.
The communication specifications limit the maximum number of binary data bytes that can be sent at once to 999999999 bytes. Therefore, this API needs to be executed several times depending on the set record length.
Note that the maximum number of binary data bytes sent by the DL950/SL2000 in a single transmission is 999999872 bytes (499999936 data points worth for voltage modules and 249999968 data points worth for RMath). (The actual number of bytes transmitted on the communication line will be greater than 999999872 bytes as supplementary information (32 bytes worth) will be included.)
The acquired waveform data is an AD value.
To convert to physical values, an appropriate data conversion is necessary according to the data type obtained with ScGetChannelType.
You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

Example [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) {
  ...
}
```

Example [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

Description:

Get the number of data points of latched waveform data in trigger mode.

Syntax:

[C++]

ScResult ScGetAcqDataLength(ScHandle hndl, int chNo, int subChNo, __int64* length);

[C#]

int ScGetAcqDataLength(int hndl, int chNo, int subChNo, out long length);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] length	Number of Data Points

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_MODE	Wrong mode error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the number of data points of the specified channel for the acquisition count specified by ScSetAcqCount.

Note:

What you can get with this API is the number of data points.
Specify the buffer size used by ScGetAcqData in bytes. The following formula is used.
buffer size = channel data's bit length × number of data points
(The channel data's bit length can be obtained with ScGetChannelBits and the number of data points with ScGetAcqDataLength.)
This API needs to be called immediately before ScGetAcqData.

You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

ScGetFreeRunDataLength

Description:

Get the number of valid points according to the time base setting in free run mode.

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] length	Number of valid points

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_MODE	Wrong mode error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the maximum number of retrievable waveform data points in free run with ScSaveFreeRunWDF.

Note:

You can use this method when SC_FREERUN mode is specified.

ScGetLatchAcqCount

Description:

Get the maximum latched acquisition count in trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] count	Maximum acquisition count at the latch point

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the maximum acquisition count at the latch point.
The obtained value is used by ScSetAcqCount.

Note:

You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

ScGetAcqCount

Description:

Get the acquisition count to be accessed in trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] count	Acquisition count to be accessed

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the acquisition count to be accessed by ScGetAcqData, ScAcqDataLength, and ScGetTriggerTime.

Note:

You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

ScSetAcqCount

Description:

Set the acquisition count to be accessed in trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] count	Acquisition count to be accessed

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Sets the acquisition count to be accessed by ScGetAcqData, ScAcqDataLength, and ScGetTriggerTime.

Note:

You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

ScGetTriggerTime

Description:

Get the trigger time

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] buff	Trigger time string

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the trigger time of the acquisition count specified by ScSetAcqCount as a string.
The time is returned as a comma separated character string.
Year (2007 or later), month (1 to 12), day (1 to 31), hour (0 to 23), minute (0 to 59), second (0 to 59), nanosecond (0 to 999999999),

Note:

You can use this method when SC_TRIGGER or SC_TRIGGER_ASYNC mode is specified.

ScResumeAcquisition

Description:

Resume waveform acquisition in synchronous trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Resumes the waveform acquisition on a DL950/SL2000 whose waveform acquisition is being held in synchronous trigger mode.

Note:

If the DL950/SL2000 is not being held in synchronous trigger mode, nothing will occur.
If the DL950/SL2000 detects a timeout, waveform acquisition will be resumed even when this command is not received.
You can use this method when SC_TRIGGER mode is specified.

ScSetTriggerTimeout

Description:

Set the timeout value on the DL950/SL2000 in synchronous trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] timeout	Timeout value (0 to 497664 s, default value: 600 s)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Sets the timeout value (in seconds) for the waveform acquisition resume command from the PC in the synchronization process with the DL950/SL2000 in synchronous trigger mode.
If set to zero, the DL950 waits until a waveform acquisition resume command is received from the PC.
If set to a value between 1 and 497664, the DL950 acquires the next waveform when the specified time elapses, without waiting for a waveform acquisition resume command from the PC.
Note that if any of the following procedures is executed before a waveform acquisition resume command is received from the PC, the timer on the DL950/SL2000 will restart.
ScGetAcqData, ScGetAcqDataLength, ScGetLatchAcqCount, ScGetAcqCount, ScSetAcqCount, ScGetTriggerTime

Note:

If the DL950 is not being held in synchronous trigger mode, nothing will occur.
You can use this method when SC_TRIGGER mode is specified.

ScGetTriggerTimeout

Description:

Get the timeout value on the DL950/SL2000 in synchronous trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] timeout	Timeout value (0 to 497664)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the timeout value (in seconds) for the waveform acquisition command from the PC in the synchronization process with the DL950/SL2000 in synchronous trigger mode.

ScGetMaxHistoryCount

Description:

Get the maximum number of histories in trigger mode

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] count	Maximum number of histories

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the maximum number of histories that can be stored in the DL950/SL2000 in trigger mode.

ScSetSamplingRate

Description:

Set the sampling frequency

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] srate	Sampling frequency (Hz)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Sets the sampling frequency.

Note:

You cannot set this during waveform acquisition.

ScGetSamplingRate

Description:

Get the sampling frequency

Syntax:

[C++]

ScResult ScGetSamplingRate(ScHandle hndl, double* srate);

[C#]

int ScGetSamplingRate(int hndl, out double srate);

Parameters:

[IN] hndl	Instrument handle
[OUT] srate	Sampling frequency

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the sampling frequency.

ScGetChannelSamplingRate

Description:

Get the channel sampling frequency

Syntax:

[C++]

ScResult ScGetChannelSamplingRate(ScHandle hndl, int chNo, int subChNo,
double* srate);

[C#]

int ScGetChannelSamplingRate(int hndlhNo, int chNo, int subChNo, out double srate);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] srate	Sampling frequency

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the channel sampling frequency.

ScGetChannelBits

Description:

Get the channel's data bit length.

Syntax:

[C++]

ScResult ScGetChannelBits(ScHandle hndl, int chNo, int subChNo, int* bits);

[C#]

int ScGetChannelBits(int hndl, int chNo, int subChNo, out int bits);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] bits	Data bit length (1 to 32)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the bit length of the channel data (valid AD values) to be acquired.

Note:

For CAN modules and the like, the returned value may not necessarily be the same as the number of bits specified with Bit Cnt.

ScGetChannelGain

Description:

Get the channel gain

Syntax:

[C++]

ScResult ScGetChannelGain(ScHandle hndl, int chNo, int subChNo, double* gain);

[C#]

int ScGetChannelGain(int hndl, int chNo, int subChNo, out double gain);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] gain	Gain

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the gain used to convert acquired waveform data into physical values.

ScGetChannelOffset

Description:

Get the channel's data offset.

Syntax:

[C++]

ScResult ScGetChannelOffset(ScHandle hndl, int chNo, int subChNo, double* offset);

[C#]

int ScGetChannelOffset(int hndl, int chNo, int subChNo, out double offset);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] offset	Offset

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the offset used to convert acquired waveform data into physical values.

ScGetChannelScale

Description:

Get the upper and lower limits of the channel display scale

Syntax:

[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);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] upper	Upper limit of display scale
[OUT] lower	Lower limit of display scale

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the upper and lower limits of the display scale set on the DL950/SL2000 screen.

ScGetChannelType

Description:

Get the channel data type

Syntax:

[C++]

ScResult ScGetChannelType(ScHandle hndl, int chNo, int subChNo, char* type);

[C#]

int ScGetChannelType(int hndl, int chNo, int subChNo, out string type);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number
[IN] subChNo	Sub channel number (specify 0 if there are none)
[OUT] type	Data type

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Parameter error

Details:

Gets the type of waveform data to be acquired as a string.

* The string is normally in abbreviated form as it conforms to the response specifications of communication commands.

ANALog	Analog format (real value = data * gain + offset)
LOGic	Logic format
FLOat	Single-precision floating-point format (applies to RMath channels)
TIMe	32-bit UNIX time and 32-bit fractional seconds in nanoseconds (applies to G5 sub channel number 63 or GPS sub channel number 7)

ScAddEventListener

Description:

Add an event listener

Syntax:

[C++]

ScResult ScAddEventListener(ScHandle hndl, ScEventListener* listener);

Parameters:

[IN] hndl	Instrument handle
[IN] listener	Pointer to the event listener class

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

A class that inherits the ScEventListener can be added as an event listener class.
The events that you can register are the over run events for free run mode and the trigger end events for synchronous trigger mode.
Overwriting handleEventScCallListener causes the same method to be called automatically when an overrun event occurs.
Overwriting handleEventScTrigEnd causes the same method to be called automatically when a trigger end event occurs.

Note:

The overrun event is valid when the connection type is not 10GEther.
This cannot be used with the .NET version (C#).
Example (free run mode):

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

Example (synchronous trigger mode):

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

ScRemoveEventListener

Description:

Delete the event listener

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] listener	Pointer to the event listener class

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Deletes a registered event listener.

Note:

An error will occur if you specify an event listener that has not been added.
This cannot be used with the .NET version (C#).

ScAddCallback

Description:

Add a call back method (C# only)

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] func	Callback method
[IN] type	Event Type

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Adds a callback method that is called when events occur.
The events that you can register are the over run events for free run mode and the trigger end events for synchronous trigger mode.
The event type will be SC_EVENTTYPE_OVERRUN or SC_EVENTTYPE_TRIGGEREND.

Note:

The overrun event is valid when the connection type is not 10GEther.
This cannot be used with C++.

Example:

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

ScRemoveCallback

Description:

Delete the call back method (C# only)

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] func	Callback method

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Deletes the call back method.

Note:

This cannot be used with C++.

4.3 Flash Acquisition Data Access Library API

ScGetFAcqCount

Description:

Get the number of flash acquisition waveform data files stored in the instrument

Syntax:

[C++]

ScResult ScGetFAcqCount(ScHandle hndl, int* count);

[C#]

int ScGetFAcqCount(int hndl, out int count);

Parameters:

[IN] hndl	Instrument handle
[OUT] count	Number of flash acquisition waveform data files stored

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the number of flash acquisition waveform data files stored in the instrument.

ScGetFAcqFileName

Description:

Get the name of a flash acquisition waveform data file stored in the instrument

Syntax:

[C++]

ScResult ScGetFAcqFileName(ScHandle hndl, int count, char* name);

[C#]

int ScGetFAcqFileName(int hndl, int count, out string name);

Parameters:

[IN] hndl	Instrument handle
[IN] count	Flash acquisition number
[OUT] name	File name

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_NODATA	No applicable files

Details:

Gets the name of a flash acquisition waveform data file stored in the instrument. Specify the flash acquisition number with the count parameter (get the number of stored files with ScGetFacqCount). The oldest flash acquisition waveform data stored is assigned to 1.

Note:

The flash acquisition number ranges from 1 to the number obtained with ScGetFacqCount.

ScOpenFAcqData

Description:

Specify the name of the flash acquisition waveform data file to open

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] name	File name
[OUT] result	Whether opening is possible

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_USE_ACQMEMORY	Waveform data exists in the acquisition memory.
SC_ERR_RUNNING	Error during measurement
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Opens a flash acquisition waveform data file to transmit to the PC.
When the file opens, zero is stored in result. Otherwise, a non-zero number is stored.
Set the file name to the name obtained with ScGetFAcqFileName.

Note:

Multiple files cannot be opened simultaneously. If you want to get multiple data files, open, close, and transmit the data one acquisition waveform data file at a time.
The acquisition memory is used as a temporary buffer when flash acquisition waveform data is transmitted. Therefore, if there is waveform data in the acquisition memory, the acquisition memory must be cleared before a file can be opened. To clear the acquisition memory, execute ScClearAcqMemory.

ScCloseFAcqData

Description:

Close the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Closes the waveform data file opened with ScOpenFAcqData.

ScClearAcqMemory

Description:

Clear the waveform data in the acquisition memory

Syntax:

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

Parameters:

[IN] hndl	Instrument handle

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Clears the waveform data in the acquisition memory.

Note:

Waveform data not stored in a storage device will be lost. Be sure to save the data as needed before executing this API command.

ScGetFAcqStartTime

Description:

Get the measurement start time and date of the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] startTime	Measurement start time

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the measurement start time as a character string for the waveform data file opened with ScOpenFAcqData.
The character string format is shown below. Fractional seconds are in unit of nanoseconds.
Output format: "2020/10/01 12:34:56:123456789"

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqTimeBase

Description:

Get the time base setting of the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] timeBase	Time base setting (0: internal, 1: external)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the time base setting of the waveform data file opened with ScOpenFAcqData.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqComment

Description:

Get the comment for the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] comment	Comment string

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the comment string for the waveform data file opened with ScOpenFAcqData.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelCount

Description:

Get the total number of channels stored in the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] count	Total number of channels

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the total number of channels stored in the opened waveform data file. Using the total number of channels obtained here, you can get the stored channel numbers with ScGetFAcqChannleNumber.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelNumber

Description:

Get the channel numbers stored in the opened waveform data file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] index	Index number
[OUT] chNo	Channel number
[OUT] subChNo	Sub Channel Number

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the channel numbers stored in the waveform data file. The channel numbers are obtained based on the specified index number. Set the index number based on the total number of channels obtained with ScGetFAcqChannelCount.
The channel numbers range from 1 to 32.
The channel numbers for RMath channels range from 17 to 32.
The channel number for the GPS channel is 17.
The channel number and sub channel number obtained here are used with the APIs for getting data and channel information.

Note:

The index number ranges from 1 to the number obtained with ScGetFacqChannelCount.
The sub channel number is 1 for modules without sub channel numbers.
This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelBits

Description:

Get the specified channel's number of bits per data point

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] bits	Number of data bits (1 to 32)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's number of bits per data point (AD value).

Note:

The number of bits here is different from the module's resolution. For example, the number of bits is 16 for channels of voltage modules and other analog modules and 32 for RMath channels. It is also 16 for logic modules (720230).
This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelGain

Description:

Get the specified channel's gain

Syntax:

[C++]

ScResult ScGetFAcqChannelGain(ScHandle hndl, int chNo, int subChNo, double* gain);

[C#]

int ScGetFAcqChannelGain(int hndl, int chNo, int subChNo, out double gain);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] gain	Sampling interval value

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's gain. The physical value of an obtained channel's data can be calculated from the gain and offset using the following formula.

Physical value = Data × Gain + Offset

Note:

The linear scale setting is applied to the gain and offset.

Conversion to physical values is necessary when the data type (ScGetFAcqChannelType) is ANALog.

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelHOffset

Description:

Gets the specified channel's offset time (seconds) from the measurement start time

Syntax:

[C++]

ScResult ScGetFAcqChannelHoffset(ScHandle hndl, int chNo, int subChNo, double* hOffset);

[C#]

int ScGetFAcqChannelHOffset(int hndl, int chNo, int subChNo, out double hOffset);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] hOffset	Offset from the measurement start time (seconds)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

The first data value when waveform acquisition was started and the offset time (phase difference) from the measurement start time can be obtained for the specified channel. The meaning of the value is as follows depending on the time base setting.
Internal sampling: Offset time from the measurement start time (seconds)
External sampling: Phase difference sample count from the measurement start point

In the case of internal sampling, when the reference sample rate (determined by the record length and T/Div settings) displayed in the upper right of the DL950/SL2000 screen differs from the channel's sample rate, a phase difference may occur between the reference sample and channel sample. In such cases, the phase difference between the data point at the start of data acquisition and the measurement start time, which represents the reference sample, for the channel is obtained as an offset time. When the channel's sample rate is the same as the reference, the value is zero.

In the case of external sampling, on modules with sub channels, external data samples, which are used as the reference, are acquired at a sample interval according to the number of sub channels. In such cases, the offset indicates the phase difference sample count between the data point at the start of channel's acquisition and the data point at the start of external sample acquisition.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelHResolution

Description:

Gets the specified channel's sampling interval

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] hResolution	Sampling interval value

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's sampling interval. The meaning of the value is as follows depending on the time base setting.
Internal sampling: Channel sampling interval (1/sampling rate) (unit: seconds)
External sampling: 1 / PulseRotate

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelLabel

Description:

Get the specified channel's label name

Syntax:

[C++]

ScResult ScGetFAcqChannelLabel(ScHandle hndl, int chNo, int subChNo, char* label);

[C#]

int ScGetFAcqChannelLabel(int hndl, int chNo, int subChNo, out string label);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] label	Channel label

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's label name.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelLogicBits

Description:

Get the specified channel's effective number of logic bits

Syntax:

[C++]

ScResult ScGetFAcqChannelLogicBits(ScHandle hndl, int chNo, int subChNo, int* bits);

[C#]

int ScGetFAcqChannelLogicBits(int hndl, int chNo, int subChNo, out int bits);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] bits	Effective number of logic bits

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's effective number of logic bits. The number is 8 when the module is 720230. On CAN modules, it is the specified number of bits.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelLogicLabel

Description:

Get the specified channel's logic bit label name

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[IN] bitNo	Bit number (1 to 8)
[OUT] label	Bit label

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's logic bit label name.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelOffset

Description:

Get the specified channel's offset value

Syntax:

[C++]

```
ScResult ScGetFAcqChannelOffset(ScHandle hndl, int chNo, int subChNo,
 double* offset);
```

[C#]

```
int ScGetFAcqChannelOffset(int hndl, int chNo, int subChNo, out double offset);
```

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] offset	Offset

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's offset. The physical value of an obtained data can be calculated from the gain and offset using the following formula.
Physical value = Data * Gain + Offset

Note:

The linear scale setting is applied to the gain and offset.
Conversion to physical values is necessary when the data type (ScGetFAcqChannelType) is ANALog.
This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelSign

Description:

Get the specified channel's data sign

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] sign	Sign (0: without a sign, 1: with a sign)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's data sign. Normal voltage modules and other analog modules are with a sign. CAN modules and other modules that have a setting for the sign are with or without a sign depending on that setting.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelType

Description:

Get the specified channel's data type

Syntax:

[C++]

ScResult ScGetFAcqChannelType(ScHandle hndl, int chNo, int subChNo, char* type);

[C#]

int ScGetFAcqChannelType(int hndl, int chNo, int subChNo, out string type);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] type	Data type

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's data type as a string.

* The string is normally in abbreviated form as it conforms to the response specifications of communication commands.

ANALog	Analog format (real value = data * gain + offset)
LOGic	Logic format
FLOat	Single-precision floating-point format (applies to RMath channels)
TIMe	32-bit UNIX time and 32-bit fractional seconds in nanoseconds (applies to G5 sub channel number 63 or GPS sub channel number 7)

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqChannelUnit

Description:

Get the specified channel's unit string

Syntax:

[C++]

ScResult ScGetFAcqChannelUnit(ScHandle hndl, int chNo, int subChNo, char* unit);

[C#]

int ScGetFAcqChannelUnit(int hndl, int chNo, int subChNo, out string unit);

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)
[OUT] unit	Unit string

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Gets the specified channel's unit string.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScSetFAcqChannelNumber

Description:

Set the channel number you want to acquire the waveform data of

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] chNo	Channel number (1 to 32)
[IN] subChNo	Sub channel number (1 to 64; specify 0 if there are none)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error
SC_ERR_PARAMETER	Parameter error

Details:

Sets the channel number you want to acquire the waveform data of. This API command applies to the ScGetFAcqDataLength and ScGetFAcqData API commands.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqDataLength

Description:

Get the specified channel's number of data points

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] length	Number of Data Points

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the number of data points for the channel specified with ScSetFacqChannelNumber.

Note:

This API command applies to the waveform data file opened with ScOpenFAcqData.

ScGetFAcqData

Description:

Get the specified channel's data

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] buff	Save buffer
[IN] buffLen	Length of save buffer (unit: bytes)
[OUT] dataLen	Length of saved data (unit: bytes)

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Gets the data for the channel specified with ScSetFacqChannelNumber. The buffer size necessary for one transmission is according to the ScSetFAcqDataSize. If the waveform data of the channel to be transmitted is larger than the buffer size, this command can be repeated consecutively to obtain the entire waveform data.
Programming tips:
You can calculate the number of loops to execute using this command from the number of data points and the number of bits or you can loop until dataLen becomes zero.

Note:

The value obtained with ScGetFAcqDataLength is the number of data points. The value used by this command is a byte-converted buffer size. To calculate the size from the number of data points, use the number of bits per point that you can get with ScGetFacqChannelBits.
If you execute this command after all the waveform data of the target channel has been acquired, the dataLen value becomes zero.
If you execute ScSetFacqChannelNumber in the middle of acquiring the waveform data of the target channel, the state will reset, and the data after the change will be transmitted from the beginning.
This API command applies to the waveform data file opened with ScOpenFAcqData.

ScSetFAcqDataSize

Description:

Set the maximum buffer size that can be transmitted at one time.

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] size	Size SC_SIZE_16MB / SC_SIZE_32MB / SC_SIZE_64MB / SC_SIZE_128MB / SC_SIZE_256MB / SC_SIZE_512MB

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_UNOPENED	File not specified
SC_ERR_PARAMETER	Size specification error
SC_ERR_NOTAPPLICABLE	Target device error

Details:

Sets the buffer size that can be transmitted from the DL950/SL2000 when ScGetFAcqData is executed once.
The available settings are 16, 32, 64, 128, 256, and 512 MiB. The default is 16 MiB.

Note:

This is the buffer size, not the number of data points.
This API command applies to the waveform data file opened with ScOpenFAcqData.

4.4 File Operation and Transfer API

ScSetCurrentDrive

Description:

Set the current drive

Syntax:

[C++]

ScResult ScSetCurrentDrive(ScHandle hndl, int drive);

[C#]

int ScSetCurrentDrive(int hndl, int drive);

Parameters:

[IN] hndl	Instrument handle	
[IN] drive	Current drive	
	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

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Current drive specification error

Details:

Sets the current drive to the specified drive.

ScGetCurrentDrive

Description:

Get the set current drive

Syntax:

[C++]

ScResult ScGetCurrentDrive(ScHandle hndl, int* drive);

[C#]

int ScGetCurrentDrive(int hndl, out int drive);

Parameters:

[IN] hndl	Instrument handle	
[OUT] drive	Current drive	
	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

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the set current drive.

ScSetCurrentDirectory

Description:

Sets the current directory.

Syntax:

[C++]

ScResult ScSetCurrentDirectory(ScHandle hndl, char* pathName);

[C#]

int ScSetCurrentDirectory(int hndl, DT[] buff, string pathName);

Parameters:

[IN] hndl	Instrument handle
[OUT] pathName	Path name
	(Example)
	"/abcdefg/efghi" → Absolute path designation
	"jklmn" → Relative path designation
	".." → To parent directory
	"/" → To root directory

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Path name empty string error

Details:

Changes the current directory.

ScGetCurrentDirectory

Description:

Gets the current directory.

Syntax:

[C++]

ScResult ScGetCurrentDirectory(ScHandle hndl, char* pathName);

[C#]

int ScGetCurrentDirectory(int hndl, out string pathName);

Parameters:

[IN] hndl	Instrument handle
[OUT] pathName	Path name
	(Example)
	"/abcdefg/efghi"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the current directory.

ScGetFileNum

Description:

Get the number of files

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[OUT] number	Number of files

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the number of files stored in the current directory.

Note:

To get file information with ScGetFileInfo, execute this API function at the target current directory in advance.

ScGetFileInfo

Description:

Get file information

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] index	File number
[OUT] info	File information

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors

Details:

Gets the information of the file with the index number in the current directory.

Note:

Get the number of files with ScGetFileNum before getting file information.

ScDeleteFile

Description:

Delete files

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] name	File name

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	File name empty string error

Details:

Deletes the file with the specified name from the current directory.

ScDownloadFile

Description:

Get a file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] srcFileName	Copy source file name (Example) "stuvw.xyz"
[IN] dstFileName	Copy destination file name (PC side) (Example) "C:\ScSDK\stuvw.xyz"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	File name empty string error

Details:

Gets the file specified by srcFileName in the current directory.

ScUploadFile

Description:

Send a file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] srcFileName	Copy source file name (PC side) (Example) "C:\ScSDK\stuvw.xyz"
[IN] dstFileName	Copy destination file name (unit side) (Example) "stuvw.xyz"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	File name empty string error

Details:

Saves the file specified by srcFileName in the current directory.

Note:

An error will occur if the file size is zero.

ScSaveTriggerWDF

Description:

Save a trigger waveform file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] dstFileName	Copy destination file name (PC side) (Example) "C:\ScSDK\stuvw.WDF"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_MODE	Wrong mode error
SC_ERR_PARAMETER	File name empty string error
SC_ERR_RUNNING	Error during measurement

Details:

Saves the displayed trigger waveform data as a WDF file in the PC.

Note:

An error will occur if a waveform is not displayed on the DL950/SL2000.
To save real-time recording data, adjust the TMCTL timeout period with ScTmcSetTimeout.
File save settings depend on the instrument settings.

ScSaveFreeRunWDF

Description:

Save a free run waveform file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] startPos	Save start position
[IN] count	Number of points to save (maximum value is the value obtained by ScGetFreeRunDataLength)
[IN] srcFileName	Copy destination file name (PC side) (Example) "C:\ScSDK\stuvw.WDF"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_MODE	Wrong mode error
SC_ERR_PARAMETER	File name empty string error
SC_ERR_RUNNING	Error during measurement

Details:

Saves the displayed free run waveform data as a WDF file in the PC.

Note:

You can get the number of valid points to be saved with ScGetFreeRunDataLength.

ScSaveSetup

Description:

Get a setup file

Syntax:

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

Parameters:

[IN] hndl	Instrument handle
[IN] fileName	Name of the setup file to be saved (PC side) (Example) "C:\ScSDK\stuvw.SET"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	File name empty string error

Details:

Saves the DL950/SL2000 settings in a setup file on the PC side.

ScLoadSetup

Description:

Set a setup file

Syntax:

[C++]

```
ScResult ScLoadSetup(ScHandle hndl, char *fileName);
```

[C#]

```
int ScLoadSetup(int hndl, string fileName);
```

Parameters:

[IN] hndl	Instrument handle
[IN] fileName	Setup file name (PC side) (Example) "C:\ScSDK\stuvw.SET"

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	File name empty string error

Details:

The settings in the setup file specified by fileName are reflected in the DL950/SL2000.

Note:

An error will occur if the file size is zero.

ScGetFileList

Description:

Get a list of file names

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle	
[OUT] fileList	List of file names	
[IN] max	Maximum size of file name list	
[IN] extension	File name extension	
	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	Sort Order	
	SC_SORT_NAME_ASC	Ascending order by name
	SC_SORT_NAME_DESC	Descending order by name
	SC_SORT_DATE_ASC	Ascending order by update date
	SC_SORT_DATE_DESC	Descending order by update date
	SC_SORT_SIZE_ASC	Ascending order by file size
	SC_SORT_SIZE_DESC	Descending order by file size
[OUT] fileCount	Number of files saved in the list	

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Maximum file name list size error, File name extension specification error

Details:

Gets a list of file names with the specified extension in the current directory.
Specify the maximum size of the array for the max parameter.

ScGetFileInfoList

Description:

Gets a list of file information

Syntax:

[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);
```

Parameters:

[IN] hndl	Instrument handle	
[OUT] fileList	List of file information	
[IN] max	Maximum size of file information list	
[IN] extension	File name extension	
	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	Sort Order	
	SC_SORT_NAME_ASC	Ascending order by name
	SC_SORT_NAME_DESC	Descending order by name
	SC_SORT_DATE_ASC	Ascending order by update date
	SC_SORT_DATE_DESC	Descending order by update date
	SC_SORT_SIZE_ASC	Ascending order by file size
	SC_SORT_SIZE_DESC	Descending order by file size
[OUT] fileCount	Number of files saved in the list	

Return value:

SC_SUCCESS	Success
SC_ERROR	Errors
SC_ERR_PARAMETER	Maximum file name list size error, File name extension specification error

Details:

Gets a list of file information with the specified extension in the current directory.
Specify the maximum size of the array for the max parameter.

4.5 DLL Linking Method

For C++, only implicit linking is currently assumed for DLL linking.
To use the API through implicit linking, specify and link to the import library (.lib file), and call the API in the same manner as calling normal functions.

In addition, place the following DLLs in the same folder as the application (exe) that you create.

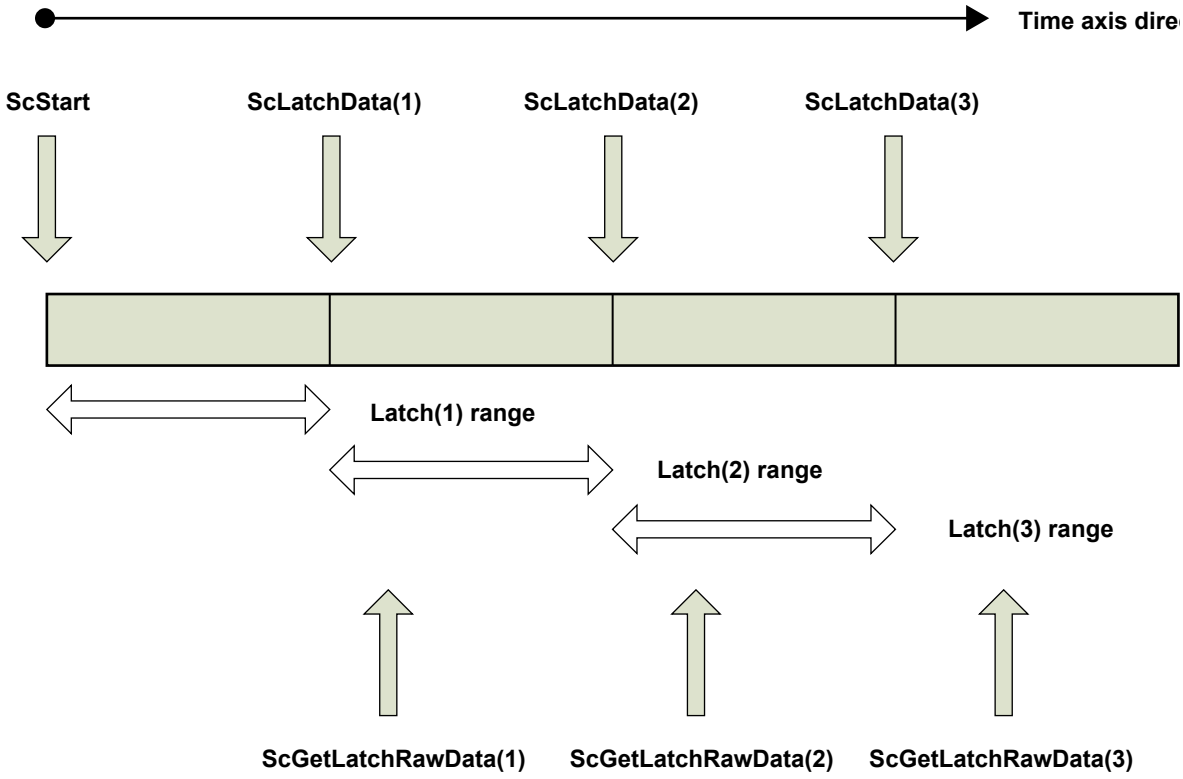
Projects	C++ (unmanaged application)		C# (managed application)		
Architecture	32bit	64bit	32bit	64bit	Any CPU
ScSDK.dll	Y		Y		Y
ScSDK 64.dll		Y		Y	Y
ScSDK Net.dll			Y	Y	Y
tmctl.dll	Y		Y		Y
tmctl64.dll		Y		Y	Y

5.1 Data Acquisition Function

Free Run Mode

Free run mode using this API and DL950/SL2000 works as follows.

The DL950/SL2000 starts acquiring waveforms when it receives a waveform acquisition start (ScStart) command. It continues to acquire waveforms until it receives a waveform acquisition stop (ScStop) command. Waveform data is temporarily stored in the instrument's acquisition memory.
While the waveform acquisition is in progress, execute latches (ScLatchData) and waveform acquisitions (ScGetLatchRawData) through the API. Waveform data between latches can be retrieved.
In a single latch, the waveform data of all channels is sent from the DL950/SL2000 to the API. Therefore, you need to be careful about the buffer size used by the API.



Sampling Rate, Wire Type, and Connection Mode

The available sampling rates for waveform acquisition vary depending on the type of connection used between the DL950/SL2000 and the API.

10G high-speed transmission

Set the write type to HiSLIP (SC_WIRE_HISLIP) when establishing a connection. In this case, the DL950/SL2000 can acquire waveforms using up to 10 MS/s × 16 channels.

Other types

If the connection is not 10G HiSLIP, the DL950/SL2000 can acquire waveforms using up to 200 kS/s × 16 channels.

If the sampling rate for acquiring waveforms is fast and the interval between data retrievals is long, waveform data in the DL950/SL2000 memory may be overwritten.

Required memory size

When data is retrieved in free run mode, the data of all waveform acquisition channels is received in the data format described in “ScGetLatchRawData Data Structure.” The required memory size must be calculated using the following parameters and set with the ScGetLatchRawData command.

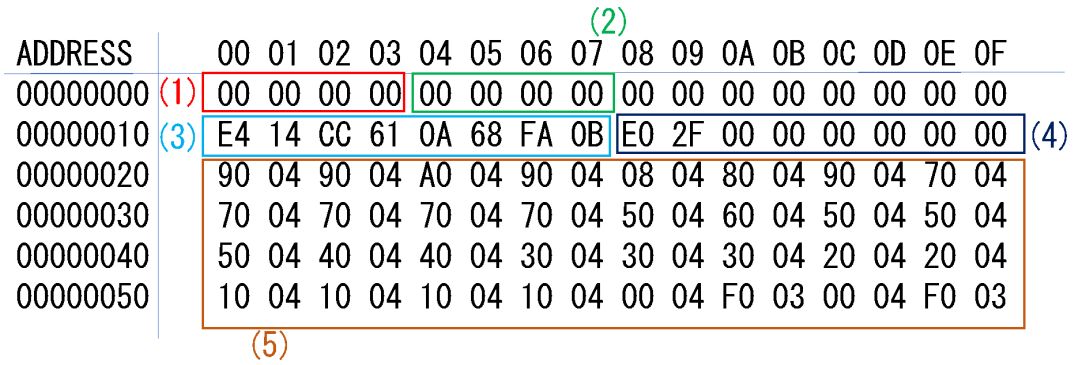
- Number of channels in use
- Sampling rate
- Latch interval

For example, if waveforms are acquired in free run mode at 200 kS/s on 16 channels (voltage module), 6400000 bytes (= 400000 bytes × 16 channels) of space are required every second.
Furthermore, 32 bytes of data are required for storing header information for each channel acquiring waveforms.
Thus, a total of 6400512 bytes (= 6400000 bytes + 32 bytes × 16 channels) of space is required every second.

ScGetLatchRawData Data Structure

In free run mode, the data received from the DL950/SL2000 contains the data of all channels acquiring waveforms.
The data format is shown below. The data of each channel is concatenated in the following format. All data is in Little Endian format.

1	Channel number (4 bytes)	0 to 31[1]	Framed area (1)
2	Sub channel number (4 bytes)	0 to 63[*1, *2]	Framed area (2)
3	Reserved (8 bytes)		
4	Time of the first data value (8 bytes)	Unix Time (4Byte) + Tick (4 bytes, in nanoseconds (0 to 999999999))[4]	Framed area (3)
5	Data size (8 bytes)	0+ The data size is equal to the number of ACQ data points converted into number of bytes. [3]	Framed area (4)
6	ACQ data	You can verify the data size of an ACQ point using ScGetChannelBits. [3]	Framed area (5)



1 Both channel numbers and sub channel numbers start at zero. (Waveform acquisition channel CH1 is '0' and RMath1 is '16'.)

2 For 720240, 720241, 720242, and 720243, the number is not the sub channel number but the number of valid sub channels.
For example, if sub channels 1 and 3 are enabled and sub channel 2 is disabled, sub channel 1 is '0' and sub channel 3 is '1'.

3 For normal modules, a single data point is 2 bytes. If 17 bits or more bytes are set on CAN, for example, a single data point is 4 bytes. For RMath channels, a single data point is 4 bytes because the data is in floating point format. For sub channels of power math and harmonic math functions, a single data point is 4 bytes because the data is in floating point format. For GPS sub channels, a single data point is 4 bytes because the data is in 32-bit integer format.
For time information channels of power math, harmonic math, and GPS functions, a single data point is 8 bytes.

4 When a measurement is performed in external sampling mode, the value of this area is undefined.

Notes for multiple sample rates and low sample rates

If waveforms are acquired at multiple sample rates or low sample rate in free run mode, the data size is adjusted so that the number of data points retrieved during waveform acquisition is fixed to a given number (integral multiple of 16). If the number becomes zero as a result of adjustment, data of the current latch is included in the data retrieved in the next latch.

Data in timestamp format

If power analysis, harmonic analysis, or GPS position information is enabled on the analysis menu, the data for these channels will be stored in timestamp format. Data in timestamp format is always stored in pairs consisting of the computed result of each item and the time information of the computation. All data is in Little Endian format.

	Power analysis		Harmonic analysis		GPS position information
Channel	RMath13	RMath14	RMath15	RMath16	RMath1
Item's sub channel	1 to 62		1 to 62		1 to 6
	32-bit floating-point type		32-bit floating-point type		32-bit integer type
Time information sub channel	63		63		7
	64-bit time format (see below)				

Time information sub channels are recorded in the following format.

4-byte data	Unix Time (with 1970/1/1 as 0)	Framed area (1)
4-byte data	Tick (4 bytes, in nanoseconds (0 to 999999999))	Framed area (2)

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)							

If the waveforms are acquired using external sampling, the sample count, not the time information, is saved.

	Sample count (64-bit counter with the first data value set to 0)		
8-byte data	4-byte data	Sample count (upper 4 bytes)	Framed area (1)
	4-byte data	Sample count (lower 4 bytes)	Framed area (2)

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)							

* The sample count is not a simple 64-bit integer value but a value divided into upper and lower bytes. Each value is in Little Endian format.

Trigger Mode

Trigger mode using this API and DL950/SL2000 works as follows.

The DL950/SL2000 acquires waveforms using triggers from when it receives a waveform acquisition start (ScStart) command until it receives a waveform acquisition stop (ScStop) command. Waveform data is stored in the instrument's acquisition memory. (The number of history entries that can be stored varies depending on the settings.)
While the waveform acquisition is in progress, execute latches (ScLatchData) and waveform acquisitions (ScGetAcqDataLength and ScGetAcqData) through the API.
There are two trigger modes: synchronous and asynchronous.

Synchronous mode

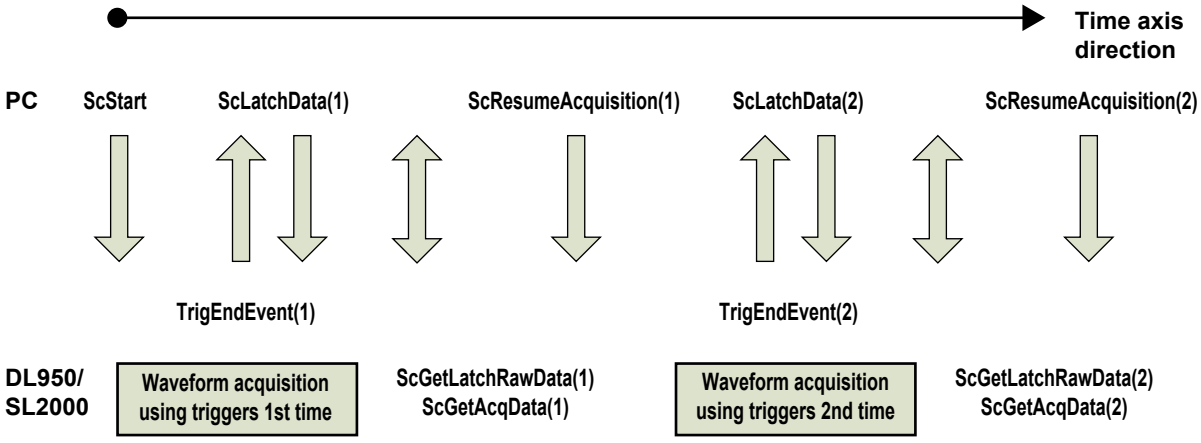
In synchronous mode, the DL950/SL2000 acquires the next waveform after waiting for a response from the PC for the previous waveform acquisition. Use synchronous mode when you want to ensure that waveform data is transferred to the PC after waveform acquisition.

There are two ways to determine the completion of a waveform acquisition on the DL950/SL2000. The first way is to implement the application to wait for trigger end events (ScAddEventListener(c++), ScAddCallBack(c#)). The second is to monitor the acquisition count (ScGetLatchAcqCount) periodically with a program and decide that a waveform acquisition has been completed when the value is updated.

* Execute ScLatchData first and then ScGetLatchAcqCount.

The following figure shows the waveform acquisition sequence using triggers with trigger end events.

Synchronous trigger mode sequence

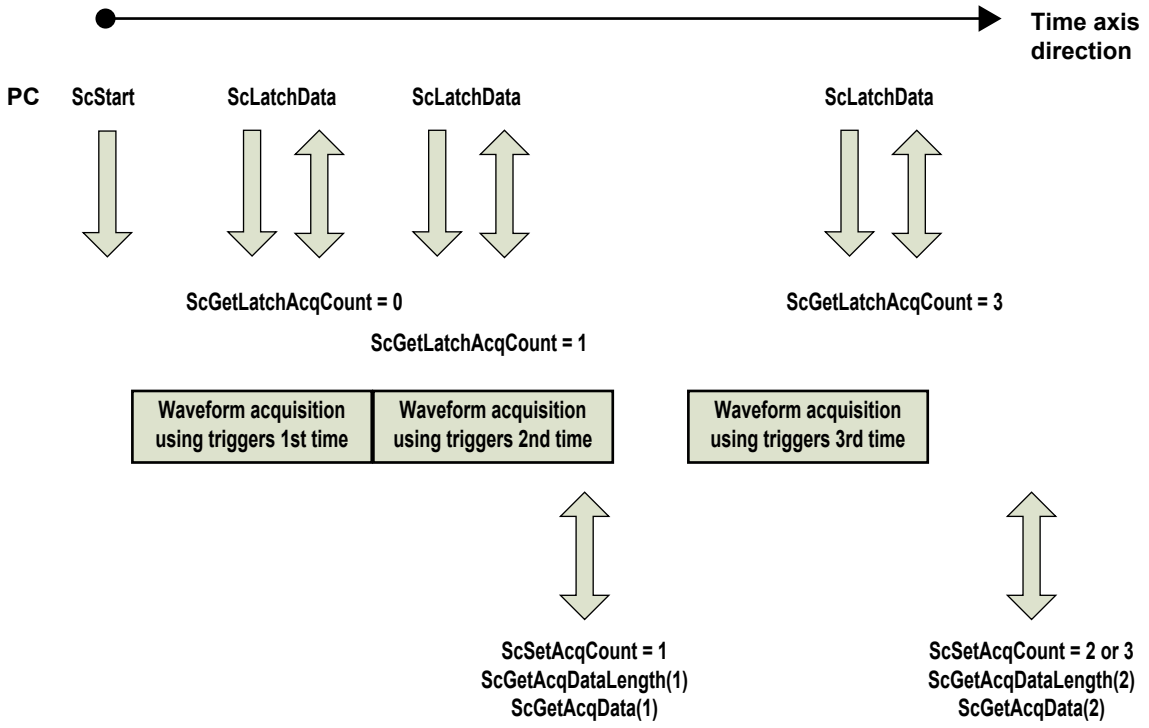


Asynchronous mode

In asynchronous mode, the DL950/SL2000 continues waveform acquisition regardless of the command control on the PC.
The PC monitors the acquisition count by periodically executing ScLatchData and ScGetLatchAcqCount. When it verifies that the acquisition count has been updated, the acquisition count sent to the PC is set using ScSetAcqCount and waveform data is transferred.
Asynchronous mode is mainly used when you want to reduce the interval (dead time) between waveform acquisitions using triggers.
In asynchronous mode, the DL950/SL2000 repeatedly acquires waveforms regardless of the waveform acquisition from the PC. If the measurement time (T/Div) is short, data may be overwritten due to waveform acquisition on the DL950/SL2000 when waveforms are acquired on the PC by specifying an old acquisition count.
(This depends greatly on the number of history entries on the DL950/SL2000. For details on the number of histories, see the appendix in the DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit User's Manual (IM DL950-02EN). In this case, waveform data cannot be acquired even when a waveform acquisition (ScGetAcqData) command is executed on the PC. The number of valid data points will be zero.

The following figure shows the waveform acquisition sequence in asynchronous mode.

Asynchronous trigger mode sequence



Waveform acquisition using external samples

When waveforms are acquired in trigger mode using external samples, timeSec and timeTick that are obtained using ScGetAcqData will contain sample counts, not time data, in the following format. (The handling is different from the timestamp format. For details on the timestamp format, see the explanation for free run mode.)

8-byte data	Sample count (64-bit counter of the first data value set to 0 after starting waveform acquisition)	
	4-byte data (timeSec)	Sample count (lower 4 bytes)
	4-byte data (timeTick)	Sample count (upper 4 bytes)

- The sample count is not a simple 64-bit integer value but a value divided into upper and lower bytes. Each value is in Little Endian format.
- The sample count is zero for the first data after waveform acquisition is started. In trigger mode, the first data obtained by this API may not necessarily be zero. (The sample count is continuously incremented until the first acquisition is completed after a trigger occurs. Therefore, the sample count at the data start point is obtained by subtracting the record length from the sample count at the point acquisition is completed.)

Further, because values are output for each calculation period for channels in timestamp format, the sample count included in the sub channel of time information in timestamp format may differ in range from the range obtained by ScGetAcqData for normal channels.

For details on the calculation period in timestamp format (power analysis), see the appendix in the *DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit Features Guide*, IM DL950-01EN.

5.2 Flash acquisition data access library

Care must be taken in handling a portion of the waveform data acquired by the DL950/SL2000. Analyze the data by referring to the following description.

Data in timestamp format

If power analysis, harmonic analysis, or GPS position information is enabled on the analysis menu, the data for these channels will be stored in timestamp format. Data in timestamp format is always stored in pairs consisting of the computed result of each item and the time information of the computation. All data is in Little Endian format.
For details on the calculation period in timestamp format (power analysis), see the appendix in the *DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit Features Guide*, IMDL950-01EN.

	Power analysis		Harmonic analysis		GPS position information
Channel	RMath13	RMath14	RMath15	RMath16	RMath1
Item's sub channel	1 to 62		1 to 62		1 to 6
	32-bit floating-point type		32-bit floating-point type		32-bit integer type
Time information sub channel	63		63		7
	64-bit time format (see below)				

Time information sub channels are recorded in the following format.

4-byte data	Unix Time (with 1970/1/1 as 0)	Framed area (1)
4-byte data	Tick (4 bytes, in nanoseconds (0 to 999999999))	Framed area (2)

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)											

If the waveforms are acquired using external sampling, the sample count, not the time information, is saved.

	Sample count (64-bit counter with the first data value set to 0)		
8-byte data	4-byte data	Sample count (upper 4 bytes)	Framed area (1)
	4-byte data	Sample count (lower 4 bytes)	Framed area (2)

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)											

* The sample count is not a simple 64-bit integer value but a value divided into upper and lower bytes. Each value is in Little Endian format.

5.3 How to Use Communication Commands

Communication commands are sent and received using the following communication command control functions.

API Name	Function	Page
ScSetControl	Send a communication command	4-7
ScGetControl	Receive a response to a communication command	4-8
ScGetBinaryData	Receive binary data	4-9
ScQueryMessage	Send a communication command and receive its response	4-10

For details on communication command control functions, see “Common API.” For details on commands, see the DL950 ScopeCorder/SL2000 High-Speed Data Acquisition Unit Communication Interface User’s Manual, IM DL950-17EN.

5.4 Migration to ScopeCorder SDK

You can smoothly migrate from DL950ACQAPI or DL950FACQAPI to ScopeCorder SDK by changing programs as follows.

C++
Remove the referenced API, add “ScSDK.lib” or “ScSDK64.lib” to the library and change the header file to “ScSDK.h.”

C#
Remove the referenced API, add “ScSDKNet” to the include statement and change the using statement to “using ScSDK = ScSDKNet.ScSDK.”

VB
Remove the referenced API, add “ScSDKNet” to the include statement and change the Imports statement to “Imports ScSDK = ScSDKNet.ScSDK.”

5.5 Comparison with the SL1000 Control API (SxAPI)

The following is a list of ScopeCorder SDK functions that have the same functionality as the SL1000 control API functions.

Function Description	SL1000 Control API	ScopeCorder SDK
Initialization	SxInit	ScInit
Close	SxExit	ScExit
Send a command	SxSetControl	ScSetControl
Send or get a command	SxGetControl	ScGetControl
		ScQueryMessage
Send or get a command	SxGetControlBinary	ScGetBinaryData
Create event handler, enable notification	SxCreateEvent	ScAddEventListener
		ScAddCallback
Delete event handler	SxDeleteEvent	ScRemoveEventListener
		ScRemoveCallback
Set the measuring mode	SxSetAcqMode	ScSetMeasuringMode
Set the sampling frequency	SxSetSamplingRate	ScSetSamplingRate
Get the sampling frequency	SxGetSamplingRate	ScGetSamplingRate
		ScGetChannelSamplingRate
Start measurement	SxAcqStart	ScStart
		ScStartEx
Stop measurement	SxAcqStop	ScStop
		ScStopEx
Execute latch	SxAcqLatch	ScLatchData
		ScLatchDataEx
Generate manual trigger	SxExecManualTrig	ScResumeAcquisition
Get acquisition data information	SxGetChannelInfo	ScGetChannelGain
		ScGetChannelOffset
Get the number of sample points in the latch interval	SxGetLatchLength	ScGetAcqDataLength
Get the latest acquisition number	SxGetLatestAcqNo	ScGetLatchAcqCount
Get waveform data	SxGetAcqData	ScGetLatchRawData
		ScGetChAcqData
Get data acquisition time	SxGetAcqTime	ScGetTriggerTime
Save setup information	SxSaveSetup	ScSaveSetup
Load setup information	SxLoadSetup	ScLoadSetup
Set the current drive	SxFileSetCurrentDrive	ScSetCurrentDrive
Get the current drive	SxFileGetCurrentDrive	ScGetCurrentDrive
Set the current directory	SxFileChDir	ScSetCurrentDirectory
Get the current directory	SxFileCwDir	ScGetCurrentDirectory
Get the number of files	SxFileGetFileNum	ScGetFileNum
Get file information	SxFileGetFileInfo	ScGetFileInfo
Delete files	SxFileDelete	ScDeleteFile
Get a file	SxFileGet	ScDownloadFile
Create a file	SxFilePut	ScUploadFile

5.6 Sample Programs

This software includes the following sample programs.

Folder name		Description
file		File operation and transfer
flashacquition		Flash acquisition data access
freerun	MultiUnit	Data acquisition free run for multi-unit synchronization
	SingleUnit	Data acquisition free run
reopen		Reconnect instruments and set measurement modes
trigger	MultiUnit	Data acquisition trigger for multi-unit synchronization
	SingleUnit	Data acquisition trigger

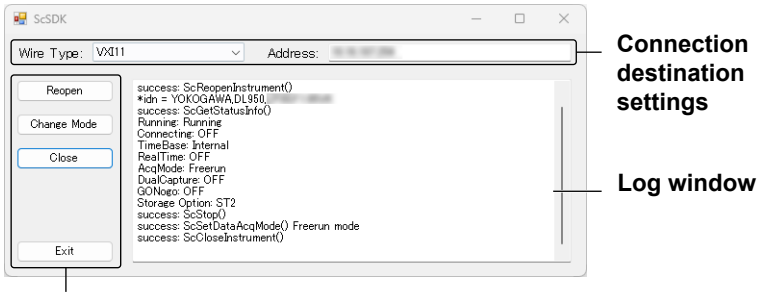
Each sample program is available in C++, C#, and VBNet.

Folder name	Description
ScSDKNetSample	Sample program (C#)
ScSDKSample	Sample program (C++)
ScSDKVBNetSample	Sample program (VBNet)

Use the APIs described in this manual by referring to these sample programs according to your environment.

Reconnect instruments and set measurement modes (reopen)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Address text box
  Enter the address of the device to be connected.

Buttons

- Reopen button
  Connects to the device under measurement that matches the selected Wire Type and the entered address. Also acquires status information and stops the measurement if it is in progress.
  API to use: ScInit, ScReopenInstrument, ScGetStatusInfo, ScStop
- Change Mode button
  Changes the connected device to free run mode.
  API to use: ScSetMeasuringMode
- Close button
  Closes the connection.
  API to use: ScCloseInstrument
- Exit button
  Closes the application.
  API to use: ScExit

Log window

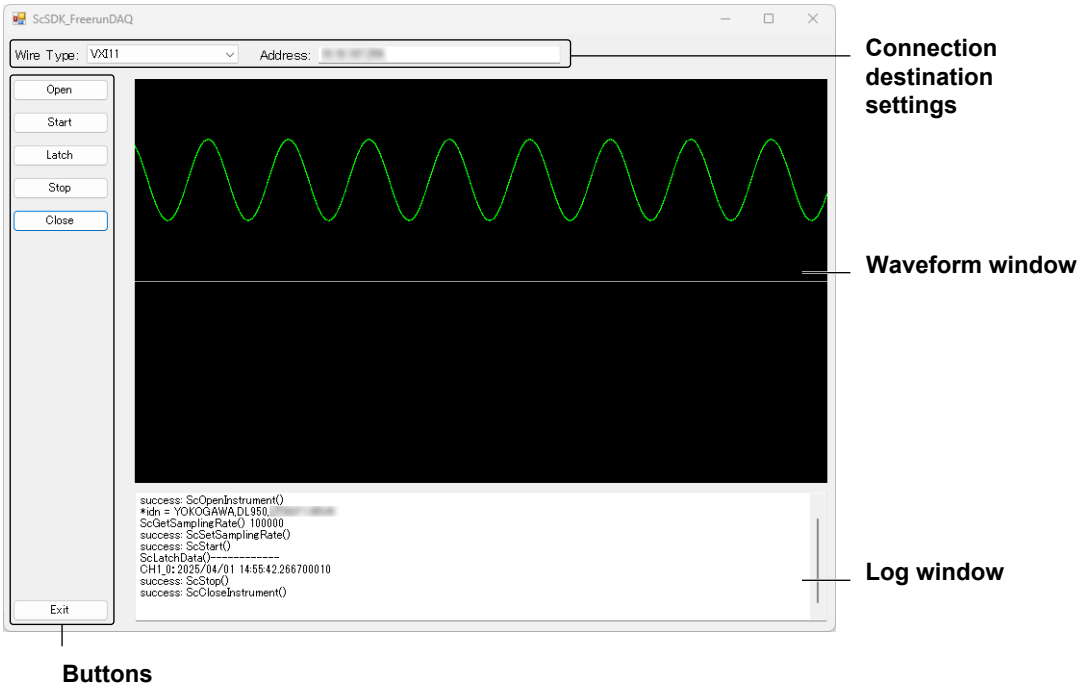
Shows connection status and data details.

How to Use

1. Specify the wire type in the Wire Type combo box and the connection destination in the Address text box.
2. Click Reopen to start a connection. The status information of the connected device is obtained, and measurement is stopped if it is in progress.
3. Click Change Mode to change the measurement mode to free run mode.
4. Click Close to close the connection.
5. Click Exit to close the application.

Data acquisition free run (freerun SingleUnit)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Address text box
  Enter the address of the device to be connected.

Buttons

- Open button
  Connects to the device that matches the selected Wire Type and the entered address.
  API to use: ScInit, ScOpenInstrument, ScGetSamplingRate
- Start button
  Starts a measurement.
  API to use: ScStart
- Latch button
  Performs a latch.
  API to use: ScLatchData, ScGetLatchRawData, ScGetChAcqData, ScGetChannelGain, ScGetChannelOffset, ScGetChannelBits, ScGetChannelType, ScGetChannelScale
- Stop button
  Stops the measurement.
  API to use: ScStop
- Close button
  Closes the connection.
  API to use: ScCloseInstrument
- Exit button
  Closes the application.
  API to use: ScExit

Waveform window

Acquires data and displays a waveform. By default, data is acquired only for CH1 and displays its waveform.

Log window

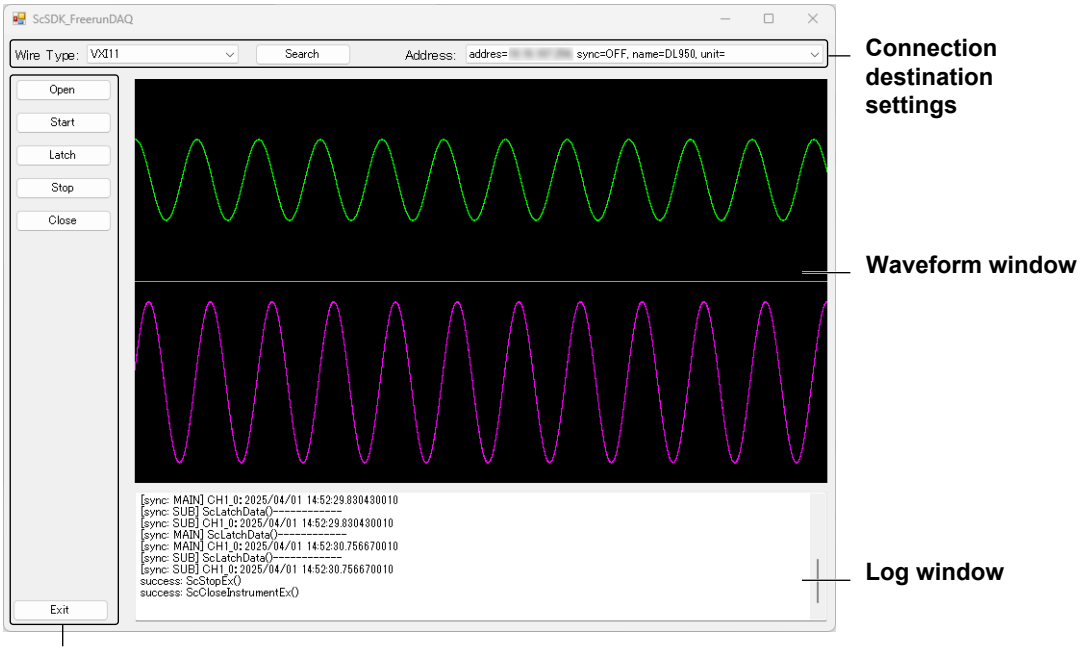
Shows connection status and data details.

How to Use

1. Specify the wire type in the Wire Type combo box and the connection destination in the Address text box.
2. Click Open to start a connection.
3. Click start to start a measurement.
4. Click Latch to acquire data in the latch period. The acquired data is displayed in the waveform window.
5. Click Stop to stop the measurement. To resume measurement, click Start.
6. Click Close to close the connection. To reconnect, click Open.
7. Click Exit to close the application.

Data acquisition free run for multi-unit synchronization (freerun MultiUnit)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Search button
  Obtains a list of instruments that can be connected with the selected wire type.
  API to use: ScInit, ScSearchDevices
- Address combo box
  Shows a list of instruments that can be connected.

Buttons

- Open button
  Connects to the device that matches the selected Wire Type and the entered address.
  API to use: ScOpenInstrumentEx, ScGetSamplingRate
- Start button
  Starts a measurement.
  API to use: ScStartEx
- Latch button
  Performs a latch.
  API to use: ScLatchDataEx, ScGetLatchRawData, ScGetChAcqData, ScGetChannelGain, ScGetChannelOffset, ScGetChannelBits, ScGetChannelType, ScGetChannelScale
- Stop button
  Stops the measurement.
  API to use: ScStopEx
- Close button
  Closes the connection.
  API to use: ScCloseInstrumentEx

- Exit button
  Closes the application.
  API to use: ScExit

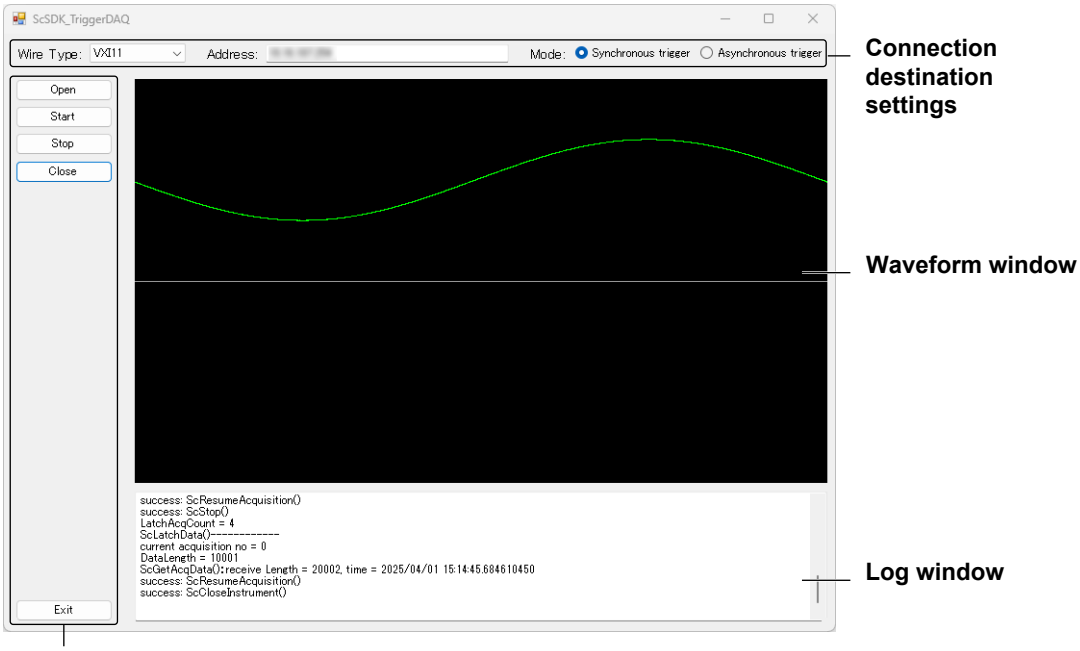
Waveform window
  Acquires data and displays a waveform. By default, data is acquired only for CH1 and displays its waveform.

Log window
  Shows connection status and data details.

How to Use
1. Select the wire type in the Wire Type combo box.
2. Click Search to obtain instruments that can be connected.
3. Select the connection destination in the Address combo box.
4. Click Open to start a connection.
5. Click start to start a measurement.
6. Click Latch to acquire data in the latch period. The acquired data is displayed in the waveform window.
7. Click Stop to stop the measurement. To restart measurement, click Start.
8. Click Close to close the connection. To reconnect, click Open.
9. Click Exit to close the application.

Data acquisition trigger (trigger SingleUnit)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Address text box
  Enter the address of the device to be connected.
- Mode radio button
  Select synchronous mode or asynchronous mode.

Buttons

- Open button
  Connects to the device that matches the selected Wire Type and the entered address.
  API to use: ScInit, ScOpenInstrument, ScAddCallback (C#, VBNet), ScAddEventListener (C++), ScGetSamplingRate, ScSetTriggerTimeout, ScGetTriggerTimeout
- Start button
  Starts a measurement.
  API to use: ScStart
- Stop button
  Stops the measurement.
  API to use: ScStop
- Close button
  Closes the connection.
  API to use: ScCloseInstrument, ScRemoveCallback (C#, VBNet), ScRemoveEventListener (C++)
- Exit button
  Closes the application.
  API to use: ScExit

Waveform window

When a trigger event is received from DL950/SL2000, waveform data is acquired, and the waveform is displayed. By default, data is acquired only for CH1 and displays its waveform.

API to use: ScLatchData, ScGetLatchAcqCount, ScSetAcqCount, ScGetAcqCount, ScGetAcqData, ScGetAcqDataLength, ScGetChannelGain, ScGetChannelOffset, ScGetChannelBits, ScGetChannelType, ScGetChannelScale, ScResumeAcquisition

Log window

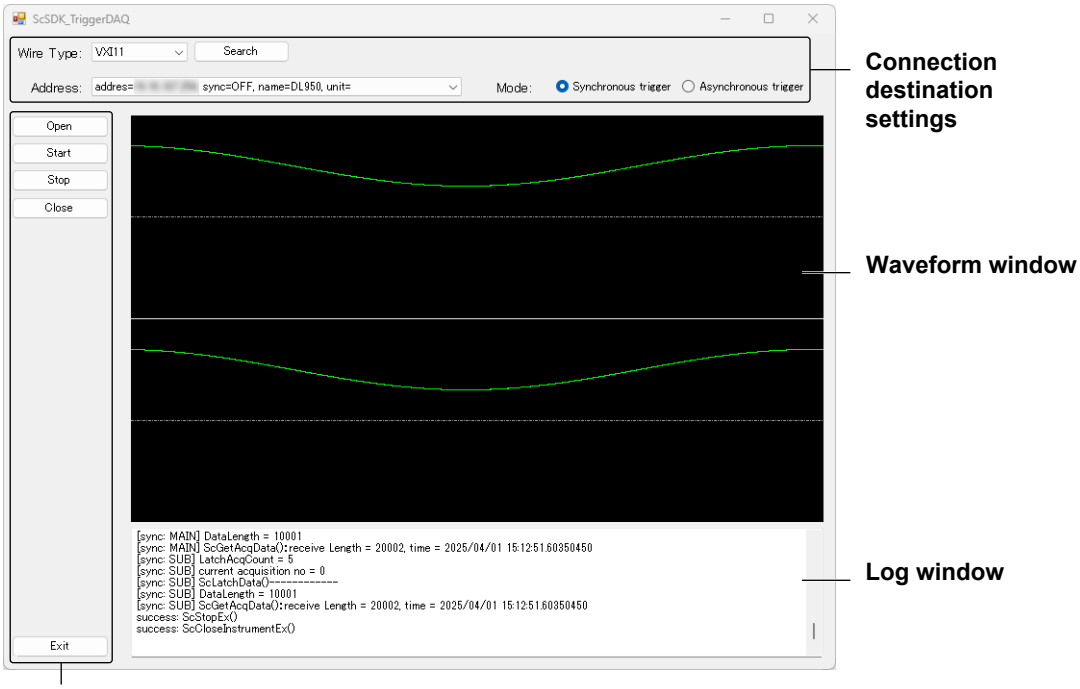
Shows connection status and data details.

How to Use

1. Specify the wire type in the Wire Type combo box and the connection destination in the Address text box.
2. Select synchronous or asynchronous mode with the Mode radio button.
3. Click Open to start a connection.
4. Click start to start a measurement. After starting, when a trigger event occurs in synchronous mode, the waveform is latched in 100 milliseconds in asynchronous mode, and the waveform is displayed on the waveform window.
5. Click Stop to stop the measurement. To restart measurement, click Start.
6. Click Close to close the connection. To reconnect, click Open.
7. Click Exit to close the application.

Data acquisition trigger for multi-unit synchronization (trigger MultiUnit)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Search button
  Obtains a list of instruments that can be connected with the selected wire type.
  API to use: ScInit, ScSearchDevices
- Address combo box
  Shows a list of instruments that can be connected
- Mode radio button
  Select synchronous mode or asynchronous mode.

Buttons

- Open button
  Connects to the selected device.
  API to use: ScOpenInstrumentEx, ScAddCallback (for C#, VBNet), ScAddEventListener (for C++), ScGetSamplingRate, ScSetTriggerTimeout, ScGetTriggerTimeout
- Start button
  Starts a measurement.
  API to use: ScStartEx
- Stop button
  Stops the measurement.
  API to use: ScStopEx

- Close button
  Closes the connection.
  API to use: ScCloseInstrumentEx, ScRemoveCallback (for C#, VBNet), ScRemoveEventListener (for C++)
- Exit button
  Closes the application.
  API to use: ScExit

Waveform window

When a trigger event is received from DL950/SL2000, waveform data is acquired, and the waveform is displayed. By default, data is acquired only for CH1 and displays its waveform.
API to use: ScLatchDataEx, ScLatchData, ScGetLatchAcqCount, ScSetAcqCount, ScGetAcqCount, ScGetAcqData, ScGetAcqDataLength, ScGetChannelGain, ScGetChannelOffset, ScGetChannelBits, ScGetChannelType, ScGetChannelScale, ScResumeAcquisition

Log window

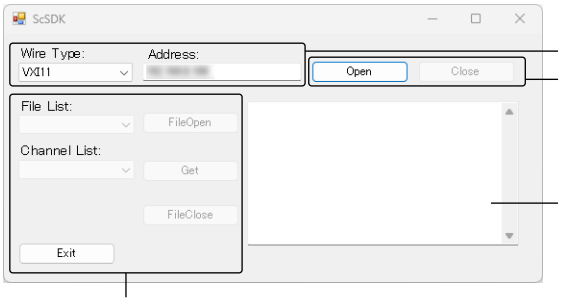
Shows connection status and data details.

How to Use

1. Select the wire type in the Wire Type combo box.
2. Click Search to obtain instruments that can be connected.
3. Specify the connection destination in the Address combo box.
4. Select synchronous or asynchronous mode with the Mode radio button.
5. Click Open to start a connection.
6. Click start to start a measurement. After starting, when a trigger event occurs in synchronous mode, the waveform is latched in 100 milliseconds in asynchronous mode, and the waveform is displayed on the waveform window.
7. Click Stop to stop the measurement. To restart measurement, click Start.
8. Click Close to close the connection. To reconnect, click Open.
9. Click Exit to close the application.

Flash acquisition data access (flashacquition)

Initial screen



Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Address text box
  Enter the address of the device to be connected.

Connection open and close buttons

- Open button
  Connects to the device that matches the selected Wire Type and the entered address.
  API to use: ScInit, ScOpenInstrument, ScGetFAcqCount, ScGetFAcqFileName
- Close button
  Closes the connection.
  API to use: ScCloseInstrument

Buttons

- File List combo box
  After the connection is started, the acquisition file name is saved in the combo box.
- FileOpen button
  Opens the acquisition file selected in the File List combo box and saves the channel list in the Channel List combo box.
  API to use: ScOpenFAcqData, ScGetFAcqChannelCount, ScGetFAcqChannelNumber, ScSetFAcqDataSize, ScSet10GMode, ScGet10GMode
- Channel List combo box
  Saves the channel list of the acquisition file.
- Get button
  Saves the channel data selected in the Channel List combo box as a CSV file.
  API to use: ScSetFAcqChannelNumber, ScGetFAcqChannelBits, ScGetFAcqDataLength, ScGetFAcqComment, ScGetFAcqChannelLabel, ScGetFAcqChannelUnit, ScGetFAcqChannelHResolution, ScGetFAcqChannelHoffset, ScGetFAcqTimeBase, ScGetFAcqStartTime, ScGetFAcqChannelGain, ScGetFAcqChannelOffset, ScGetFAcqData
- FileClose button
  Closes the acquisition file.
  API to use: ScCloseFAcqData
- Exit button
  Closes the application.
  API to use: ScExit

Log window
Shows connection status and data details.

How to Use
1. Specify the wire type in the Wire Type combo box and the connection destination in the Address text box.
2. Click Open to start a connection. A list of acquisition files of the connection destination is shown in the File List combo box.
3. Select an acquisition file in the File List combo box, and press FileOpen. A list of channels of the acquisition file is shown in the Channel List combo box.
4. Select a channel in the Channel List combo box, and press the Get to acquire the channel data.
5. Click FileClose to close the acquisition file.
6. Click Close to close the connection. To reconnect, click Open.
7. Click Exit to close the application.

File operation and transfer

Initial screen

Connection destination settings

- Wire Type combo box
  Select the wire type from USBTMC, VXI11, and HiSLIP.
- Address text box
  Enter the address of the device to be connected.

Buttons

- Open button
  Connects to the device that matches the selected Wire Type and the entered address.
  API to use: ScInit, ScOpenInstrument
- Get File List button
  Gets a list of files in the current directory and displays them in the combo box.
  API to use: ScGetFileList
- Download File button
  Gets the file selected in the combo box.
  API to use: ScDownloadFile
- Upload File button
  Saves the file you just retrieved to the device.
  API to use: ScUploadFile
- Save WDF File button
  Gets the waveform displayed in the window as a WDF file.
  API to use: ScSaveTriggerWDF
- Save Setup File button
  Gets the device settings as a SET file.
  API to use: ScSaveSetup
- Load Setup File button
  Applies the settings in the SET file you just obtained to the device settings.
  API to use: ScLoadSetup
- Close button
  Closes the connection.
  API to use: ScCloseInstrument
- Exit button
  Closes the application.
  API to use: ScExit

Log window
Shows connection status and data details.

How to Use

1. Specify the wire type in the Wire Type combo box and the connection destination in the Address text box.
2. Click Open to start a connection.
3. Press Get File List to get a list of files in the current directory. The obtained file list is shown in the combo box.
4. Select a file in the combo box.
5. Click Download to get the file selected in the combo box.
6. Click Upload File to save the file you just retrieved to the device.
7. Click Save WDF File to save the waveform displayed on the device screen as a WDF file. An error will occur if there is no waveform on the screen.
8. Click Save Setup File to get the device settings as a SET file.
9. Click Load Setup File to apply the settings in the SET file you just obtained to the device settings.
10. Click Close to close the connection. To reconnect, click Open.
11. Click Exit to close the application.