Thank you for purchasing the SL1000 Control API. The SL1000 Control API (SxAPI) is an API (application programming interface) library for creating measurement control application programs for the SL1000 High-Speed Data Acquisition Unit.

SxAPI is provided through a Microsoft Windows dynamic link library (DLL). You can use it in WIN32 development environments such as Microsoft Visual C++ and Microsoft Visual Basic, or in Microsoft .NET Framework development environments such as Microsoft Visual C# and Microsoft Visual Basic 2005. This user's manual explains the specifications of the SL1000 Control API interface. Keep this manual in a safe place for quick reference in the event a question arises.

List of Manuals

The following manuals, including this one, are provided as manuals for the SL1000. Please read all of them.

<table>
<thead>
<tr>
<th>Manual Title</th>
<th>Manual No.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL1000 High-Speed Data Acquisition Unit User’s Manual</td>
<td>IM 720120-01E</td>
<td>Explains how to install the SL1000 and its input modules, and explains features related to the hardware, such as the display, and how to operate them.</td>
</tr>
<tr>
<td>SL1000 Acquisition Software User’s Manual</td>
<td>IM 720120-61E</td>
<td>Explains all functions and procedures of the Acquisition Software used to configure and control the SL1000.</td>
</tr>
<tr>
<td>SL1000 Input Module User’s Manual</td>
<td>IM 720120-51E</td>
<td>Explains the specifications of the input modules that can be installed in the SL1000.</td>
</tr>
<tr>
<td>701992 Xviewer User’s Manual</td>
<td>IM 701992-01E</td>
<td>Explains all functions and procedures of the Xviewer software used to display the measured data as waveforms on a PC. This manual is not included with the /XV0 option.</td>
</tr>
<tr>
<td>SL1000 Control API User’s Manual</td>
<td>IM 720320-01E</td>
<td>This manual. It explains the functions for controlling the SL1000 (the SL1000 control API).</td>
</tr>
<tr>
<td>SL1000 High-Speed Data Acquisition Unit Communication Interface User’s Manual</td>
<td>IM 720320-17E</td>
<td>Explains the communication interface functions of the SL1000.</td>
</tr>
</tbody>
</table>

Notes

- The contents of this manual apply to the SL1000 Control API Ver. 3.00. If you use another version of the SL1000 Control API, its contents may be different than those of the Control API described in this manual.
- The contents of this manual are subject to change without prior notice as a result of continuing improvements to the instrument’s performance and functionality. 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, Active X, Visual Basic, Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated.
- 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: May 2008
- 3rd Edition: September 2013
- 4th Edition: January 2015
- 5th Edition: April 2019
Yokogawa Electric Corporation and Yokogawa Test & Measurement Corporation, Japanese corporations (hereinafter called “Yokogawa”), grants permission to use this Yokogawa Software Program (hereinafter called the “Licensed Software”) to the Licensee on the conditions that the Licensee agrees to the terms and conditions stipulated in Article 1 hereof.

You, as the Licensee (hereinafter called “Licensee”), shall agree to the following terms and conditions for the software license (hereinafter called the “Agreement”) based on the use intended for the Licensed Software.

Please note that Yokogawa grants the Licensee permission to use the Licensed Software under the terms and conditions herein and in no event shall Yokogawa intend to sell or transfer the Licensed Software to the Licensee.

Licensed Software Name: SL1000 Control API
Number of License: 1

Article 1 (Scope Covered by these Terms and Conditions)
1.1 The terms and conditions stipulated herein shall be applied to any Licensee who purchases the Licensed Software on the condition that the Licensee consents to agree to the terms and conditions stipulated herein.

1.2 The “Licensed Software” herein shall mean and include all applicable programs and documentation, without limitation, all proprietary technology, algorithms, and know-how such as a factor, invariant or process contained therein.

Article 2 (Grant of License)
2.1 Yokogawa grants the Licensee, for the purpose of single use, non-exclusive and non-transferable license of the Licensed Software with the license fee separately agreed upon by both parties.

2.2 Yokogawa shall otherwise agree in writing by Yokogawa, not entitled to copy, change, sell, distribute, transfer, or sublicense the Licensed Software.

2.3 The Licensed Software shall not be copied in whole or in part except for keeping one (1) copy for back-up purposes. The Licensee shall secure or supervise the copy of the Licensed Software by the Licensee itself with strict, and due care.

2.4 In no event shall the Licensee dump, reverse assemble, reverse compile, or reverse engineer the Licensed Software so that the Licensee may translate the Licensed Software into other programs or change it into a man-readable form from the source code of the Licensed Software. Unless otherwise separately agreed by Yokogawa, Yokogawa shall not provide the Licensee the source code for the Licensed Software.

2.5 The Licensed Software and its related documentation shall be the proprietary property or trade secret of Yokogawa or a third party which grants Yokogawa the rights. In no event shall the Licensed Software be transferred, leased, sublicensed, or assigned any rights relating to the Licensed Software.

2.6 Yokogawa may use or add copy protection in or onto the Licensed Software. In no event shall the Licensee remove or attempt to remove such copy protection.

2.7 The Licensed Software may include a software program licensed for re-use by a third party (hereinafter called “Third Party Software”, which may include any software program from affiliates of Yokogawa made or coded by themselves). In the case that Yokogawa is granted permission to sublicense to third parties by any licensors (sub-licensors) of the Third Party Software to grant to different terms and conditions than those stipulated in this Agreement, the Licensee shall observe such terms and conditions of which Yokogawa notifies the Licensee in writing separately.

2.8 In no event shall the Licensee modify, remove or delete a copyright notice of Yokogawa and its licensor contained in the Licensed Software, including any copyright thereof.

Article 3 (Restriction of Specific Use)
3.1 The Licensed Software shall not be intended specifically to be designed, developed, constructed, manufactured, distributed or maintained for the purpose of the following events:

- o) Operation of any aviation, vessel, or support of those operations from the ground,
- b) Operation of nuclear products and/or facilities,
- c) Operation of nuclear weapons and/or chemical weapons and/or biological weapons, or
- d) Operation of medical instrumentation directly utilized for humankind or the human body.

3.2 Even if the Licensee uses the Licensed Software for the purposes in the preceding Paragraph 3.1, Yokogawa has no liability or responsibility for any demand or damage arising out of the use of operations of the Licensed Software, and the Licensee agrees, on its own responsibility, to solve and settle the claims and damages and to defend, indemnify or hold Yokogawa totally harmless, from or against any liabilities, losses, damages and expenses (including fees for recalling the Products and reasonable attorney's fees and court costs), or claims arising out of and related to the above-said claims and damages.

Article 4 (Warranty)
4.1 The Licensee shall agree that the Licensed Software shall be provided to the Licensee on an “as is” basis when delivered. If defect(s), such as damage to the medium of the Licensed Software, attributable to Yokogawa is found, Yokogawa agrees to, free of charge, any Licensed Software on condition that the defective Licensed Software shall be returned to Yokogawa’s specified authorized service facility within seven (7) days after opening the Package at the Licensee’s expense. As the Licensed Software is provided to the Licensee on an “as is” basis when delivered, in no event shall Yokogawa warrant that any information on or in the Licensed Software, including without limitation, any computer programs and program listings, be completely accurate, correct, reliable, or the most updated.

4.2 Notwithstanding the preceding Paragraph 4.1, when third party software is included in the Licensed Software, the warranty period and terms and conditions that apply shall be those established by the provider of the third party software.

4.3 When Yokogawa decides in its own judgment that it is necessary, Yokogawa may from time to time provide the Licensee with Revision upgrades and Version upgrades separately specified by Yokogawa (hereinafter called “Updates”).

4.4 Notwithstanding the preceding Paragraph 4.3, in no event shall Yokogawa provide Updates where the Licensee or any third party conducted renovation or improvement of the Licensed Software.

4.5 THE FOREGOING WARRANTIES ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES OF QUALITY AND PERFORMANCE, WRITTEN, ORAL, IMPLIED, AND ALL OTHER WARRANTIES INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED BY YOKOGAWA AND ALL THIRD PARTIES LICENSING THIRD PARTY SOFTWARE TO YOKOGAWA.

4.6 The correction of nonconformity in the manner and for the period of time provided above shall be the Licensee’s sole and exclusive remedy for any failure of Yokogawa to comply with its obligations and shall constitute fulfillment of all liabilities of Yokogawa and any third party licensing the Third Party Software to Yokogawa (including any liability for direct, indirect, special, incidental or consequential damages) whether in warranty, contract, tort (including negligence but excluding willful conduct or gross negligence by Yokogawa) or otherwise with respect to or arising out of the use of the Licensed Software.

Article 5 (Infringement)
5.1 If and when any third party should demand injunction, initiate a law suit, or demand compensation for damages against the Licensee under patent right (including utility model right, design patent, and trade mark), copy right, and any other rights relating to any of the Licensed Software, the Licensee shall notify Yokogawa in writing to that effect without delay.

5.2 In the case of the preceding Paragraph 5.1, the Licensee shall assign to Yokogawa all of the rights to defend the Licensee and to negotiate with the claiming party. Furthermore, the Licensee shall provide Yokogawa with written notice of any other assistance for Yokogawa’s defense and negotiation. If and when such a claim should be attributable to Yokogawa, subject to the written notice received by Yokogawa in the preceding Paragraph 5.1, Yokogawa shall defend the Licensee and negotiate with the claiming party at Yokogawa’s cost and expense and be responsible for the final settlement or judgment granted to the claiming party in the preceding Paragraph 5.1.

5.3 When any assertion or allegation of the infringement of the third party’s rights defined in Paragraph 5.1 is made, or when at Yokogawa’s judgment there is possibility of such assertion or allegation, Yokogawa will, at its own discretion, take any of the following countermeasures at Yokogawa’s cost and expense.

- a) To acquire the necessary right from the third party which has lawful ownership of the right so that the Licensee will be able to continue to use the Licensed Software;
- b) To replace the Licensed Software with an alternative one which avoids the infringement; or
- c) To remodel the Licensed Software so that the Licensed Software can avoid the infringement of such third party’s right.

5.4 If and when Yokogawa fails to take either of the countermeasures as set forth in the preceding subparagraphs of Paragraph 5.3, Yokogawa shall indemnify the Licensee only by paying back the price amount of the Licensed Software which Yokogawa has received from the Licensee.

THE FOREGOING PARAGRAPHS STATE THE ENTIRE LIABILITY OF YOKOGAWA AND ANY THIRD PARTY LICENSED TO YOKOGAWA WITH RESPECT TO INFRINGEMENT OF THE INTELLECTUAL PROPERTY RIGHTS INCLUDING BUT NOT LIMITED TO, PATENT AND COPYRIGHT.
Article 6 (Liabilities)
6.1 If and when the Licensee should incur any damage relating to or arising out of the Licensed Software or service that Yokogawa has provided to the Licensee under the conditions herein due to a reason attributable to Yokogawa, Yokogawa shall take actions in accordance with this Agreement. However, in no event shall Yokogawa be liable or responsible for any special, incidental, consequential and/or indirect damage, whether in contract, warranty, tort, negligence, strict liability, or otherwise, including, without limitation, loss of operational profit or revenue, loss of use of the Licensed Software, or any associated products or equipment, cost of capital, loss or cost of interruption of the Licensee’s business, substitute equipment, facilities or services, downtime costs, delays, and loss of business information, or claims of customers of Licensee or other third parties for such or other damages. Even if Yokogawa is liable or responsible for the damages attributable to Yokogawa and to the extent of this Article 6, Yokogawa’s liability for the Licensee’s damage shall not exceed the price amount of the Licensed Software or service fee which Yokogawa has received. Please note that Yokogawa shall be released or discharged from part or all of the liability under this Agreement if the Licensee modifies, remodels, combines with other software or products, or causes any deviation from the basic specifications or functional specifications, without Yokogawa’s prior written consent.

6.2 All causes of action against Yokogawa arising out of or relating to this Agreement or the performance or breach hereof shall expire unless Yokogawa is notified of the claim within one (1) year of its occurrence.

6.3 In no event, regardless of cause, shall Yokogawa assume responsibility for or be liable for penalties or penalty clauses in any contracts between the Licensee and its customers.

Article 7 (Limit of Export)
Unless otherwise agreed by Yokogawa, the Licensee shall not directly or indirectly export or transfer the Licensed Software to any countries other than those where Yokogawa permits export in advance.

Article 8 (Term)
This Agreement shall become effective on the date when the Licensee receives the Licensed Software and continues in effect unless or until terminated as provided herein, or the Licensee ceases using the Licensed Software by itself or with Yokogawa’s thirty (30) days prior written notice to the Licensee.

Article 9 (Injunction for Use)
During the term of this Agreement, Yokogawa may, at its own discretion, demand injunction against the Licensee in case that Yokogawa deems that the Licensed Software is used improperly or under severer environments other than those where Yokogawa has first approved, or any other condition which Yokogawa may not permit.

Article 10 (Termination)
Yokogawa, at its sole discretion, may terminate this Agreement without any notice or reminder to the Licensee if the Licensee violates or fails to perform this Agreement. However, Articles 5, 6, and 11 shall survive even after the termination.

Article 11 (Jurisdiction)
Any dispute, controversies, or differences between the parties hereto as to interpretation or execution of this Agreement shall be resolved amicably through negotiation between the parties upon the basis of mutual trust. Should the parties fail to agree within ninety (90) days after notice from one of the parties to the other, both parties hereby irrevocably submit to the exclusive jurisdiction of the Tokyo District Court (main office) in Japan for settlement of the dispute.

Article 12 (Governing Law)
This Agreement shall be governed by and construed in accordance with the laws of Japan. The Licensee expressly agrees to waive absolutely and irrevocably and to the fullest extent permissible under applicable law any rights against the laws of Japan which it may have pursuant to the Licensee’s local law.

Article 13 (Severability)
In the event that any provision hereof is declared or found to be illegal by any court or tribunal of competent jurisdiction, such provision shall be null and void with respect to the jurisdiction of that court or tribunal and all the remaining provisions hereof shall remain in full force and effect.
# How to Use This Manual

## Structure of the Manual

This manual contains five chapters, an appendix, and an index.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General Features</td>
<td>Explains the general features of SxAPI.</td>
</tr>
<tr>
<td>2</td>
<td>Events</td>
<td>Explains event conditions and programming.</td>
</tr>
<tr>
<td>3</td>
<td>Function Details</td>
<td>Explains the details of all of the SxAPI functions.</td>
</tr>
<tr>
<td>4</td>
<td>SL1000 Communication Commands</td>
<td>Explains the communication commands that the SL1000 supports.</td>
</tr>
<tr>
<td>5</td>
<td>Error Codes</td>
<td>Explains SxAPI error codes and SL1000 error codes.</td>
</tr>
</tbody>
</table>

**Appendix**

Contains sample programs.

**Index**
Contents

List of Manuals...................................................................................................................................i
Terms and Conditions of the Software License................................................................................ iii
How to Use This Manual......................................................................................................................v

Chapter 1 General Features
1.1 File Structure and Operating Environment................................................................. 1-1
1.2 Programming with SxAPI........................................................................................ 1-3
1.3 Programming Flowchart ......................................................................................... 1-5
1.4 .NET Control Library............................................................................................... 1-6

Chapter 2 Events
2.1 Event Conditions ........................................................................................................ 2-1
2.2 Programming Methods ............................................................................................ 2-2

Chapter 3 Function Details
3.1 Functions .................................................................................................................. 3-1
3.2 Initialization and Ending .......................................................................................... 3-6
3.3 Retrieving Device Information ............................................................................... 3-8
3.4 Opening and Closing Handles ................................................................................ 3-10
3.5 Handle Acquisition .................................................................................................. 3-12
3.6 Value Retrieval ........................................................................................................ 3-15
3.7 Measuring Group Settings ...................................................................................... 3-17
3.8 Communication Command Controls ....................................................................... 3-18
3.9 Event Controls ......................................................................................................... 3-23
3.10 Measurement Condition Settings and Queries ...................................................... 3-25
3.11 Auto Recording Condition Settings and Queries .................................................. 3-34
3.12 Measurement Controls .......................................................................................... 3-44
3.13 Auto Recording Controls ....................................................................................... 3-46
3.14 Acquisition and Deletion of Measured Data ............................................................ 3-47
3.15 Setup Data Access ................................................................................................... 3-54
3.16 System-Related Functions ...................................................................................... 3-55
3.17 Internal Media Operations ..................................................................................... 3-62
3.18 Debugging ................................................................................................................ 3-68
3.19 Definitions .............................................................................................................. 3-71

Chapter 4 SL1000 Communication Commands
4.1 Using Communication Commands ........................................................................... 4-1
4.2 Commands ................................................................................................................ 4-2
4.3 ALARm Group ......................................................................................................... 4-6
4.4 CHAnnel Group ...................................................................................................... 4-9
4.5 GONogo Group ....................................................................................................... 4-20
4.6 MEASure Group ..................................................................................................... 4-22
4.7 TRIGger Group ...................................................................................................... 4-24
Chapter 5 Error Codes
5.1 Library Errors .................................................................................................................. 5-1
5.2 Unit Errors ........................................................................................................................ 5-2

Appendix
Appendix 1 Sample Programs ............................................................................................. App-1

Index
### 1.1 File Structure and Operating Environment

#### Folder and File Structure

<table>
<thead>
<tr>
<th>Folder</th>
<th>File Name</th>
<th>Redistribution</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL1000</td>
<td>SxAPI.dll</td>
<td>Yes</td>
<td>The dynamic link libraries necessary to execute a program. Put them in the same folder as the executable program (.exe) files.</td>
</tr>
<tr>
<td></td>
<td>tmctl.dll</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>YKMUSB.dll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIB</td>
<td>SxAPI.h</td>
<td>No</td>
<td>The header and library files that are necessary when programming in C or C++.</td>
</tr>
<tr>
<td>VB6</td>
<td>SxAPI.NET.dll</td>
<td>Yes</td>
<td>The .NET control library necessary for programming or executing VB.NET or VC# programs. Put it in the same folder as the executable program (.exe) files.</td>
</tr>
<tr>
<td></td>
<td>SxEvent.ocx</td>
<td>Yes</td>
<td>The ActiveX control used to process SxAPI asynchronous messages in VB6.</td>
</tr>
<tr>
<td>x64</td>
<td>SxAPI.dll</td>
<td>Yes</td>
<td>A library for the 64-bit version.</td>
</tr>
<tr>
<td></td>
<td>tmctl64.dll</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>YKMUSB64.dll</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SxAPI.h</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SxAPI.lib</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SxAPI.NET.dll1</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

This package does not include a USB driver.

To connect through USB, install a USB driver according to the instructions in the acquisition software manual supplied with the SL1000 unit.

1: If you want to program using VB.NET or VC#, use the 32-bit SxAPI.NET.dll or the 64-bit SxAPI.NET.dll (in the x64 folder) depending on the platform target specified in the project.

In addition, place the 32-bit SxAPI.NET.dll, tmctl.dll, and YKMUSB.dll or the 64-bit SxAPI.NET.dll, tmctl64.dll, and YKMUSB64.dll in the same folder as the executable.

### Operating Environment

#### OS


#### Development Environment

Microsoft Visual C++ 2008 or later, Microsoft Visual Basic 6.0, Microsoft Visual Basic 2008 or later, or Microsoft Visual C# 2008 or later
1.1 File Structure and Operating Environment

Position

The diagram below shows the position of the API in relation to an application program. The grey area contains software that the API provides.
1.2 Programming with SxAPI

Handles

SxAPI is a handle-based API.

In general, a file is accessed when its file name is specified and the file is opened. Then, a file handle is obtained that is used for reading and writing data. Communication types and group IDs are specified and opened (connected to) in the same way. The communication handles and unit group handles that are obtained are used to acquire information, make measurement settings, and control execution.

To make unit, module, and channel access easy, SxAPI handles have the following features.

1. A unit group composed of multiple units can be specified with a single handle (a unit group handle). Because of this, through the execution of a single function, the functions in the same unit group can be accessed.

2. Unit handles, module handles, and channel handles can be acquired from unit group handles. Through the use of different handles for different settings, the appropriate items can be accessed efficiently.

3. Properties such as communication types, group IDs, slot numbers, and channel numbers are managed and processed logically, so programs are not effected by the network connection location or the module installation slot. This makes it possible to develop very stable programs.

The kinds of handles are listed below.

<table>
<thead>
<tr>
<th>Handle Type and Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication handle SX_HNDL_COMM</td>
<td>This handle is acquired by using SxInit() to establish a connection. It handles communication. Be sure to disconnect by using SxExit() before closing a program.</td>
</tr>
<tr>
<td>Unit group handle SX_HNDL_GROUP</td>
<td>This handle is acquired by using SxOpenGroup() to open a unit group. It handles the open unit group. Be sure to close the unit group by using SxCloseGroup() before disconnecting.</td>
</tr>
<tr>
<td>Unit handle SX_HNDL_UNIT</td>
<td>This handle is acquired by using SxUnitHndl() or SxMyUnitHndl(). It handles a single unit. There is no close operation.</td>
</tr>
<tr>
<td>Module handle SX_HNDL_MOD</td>
<td>This handle is acquired by using SxModHndl() or SxMyModHndl(). It handles a single module. There is no close operation.</td>
</tr>
<tr>
<td>Channel handle SX_HNDL_CH</td>
<td>This handle is acquired by using SxChHndl(). It handles a single channel. There is no close operation.</td>
</tr>
<tr>
<td>Measuring group handle SX_HNDL_MEASGRP</td>
<td>This handle is acquired by using SxMeasgrpHndl() or SxMyMeasgrpHndl(). It handles a single measuring group. There is no close operation.</td>
</tr>
</tbody>
</table>

The relationships between handles are illustrated in a tree structure on the next page.
Settings and Queries

In addition to handle control, the SxAPI offers functions for setting time-axis-related measurement conditions such as the measuring mode, sampling rate, and recording time, for querying, and for acquiring and saving measured data. There are settings that relate to the vertical axis, such as the voltage range, coupling, and trigger level, that are available depending on the module. When using these settings and settings that are not provided through specialized functions, such as alarm, waveform parameter computation, and GO/NO-GO judgment, send and receive unit communication commands through API commands such as SxSetControl() and SxGetControl(). For unit communication command specifications, see chapter 4.

In this API, the reply to unit query commands is automatically set to “data only” (in other words, “no header”) when a unit is opened.

Asynchronous Messages (Events)

To increase the efficiency of application program execution, SxAPI can indicate trigger detection and measurement completion with window messages. These indications are referred to as events. Events are produced using unit SRQ interrupts. This allows you to write programs that run based on received events. Also, to make this kind of message processing easy, we have also prepared an OCX that can be used in Visual Basic 6.0. In the .NET control library, SRQ notification takes place through the event method.

For details, see chapter 2.
1.3 Programming Flowchart

The following diagram is a basic flowchart of an application program that uses SxAPI.

Start program

Connection
Sxht()

Open unit group
SxOpenGroup()

Create event handler
SxCreateEvent()

Set/clear measurement conditions
SxSetControl()
SxGetControl()
etc.

Start measurement
SxAcqStart()

Wait for event

Delete event handler
SxCreateEvent()

Close unit group
SxCloseGroup()

Disconnect
SxExit()

Close program

Event handler

End measurement notification?

Y

Save/load measured data
SxSaveAcqData()
SxGetAcqData()
etc.

Event handler

N
1.4 .NET Control Library

For programming in a .NET environment, the SxAPI.NET.dll control library is used. The control library specifications are as follows.

Namespace
namespace SxAPI

Classes
The following two classes are available:
class SxAPI   Provides all functions as methods.
class SxEventArgs  Is a delegate class that provides event parameters to the Event method.

Methods
All functions are provided as SxAPI class methods.
In Chapter 3, “Function Details,” .NET method interfaces are printed in italics.
As a rule, the method that corresponds to a VC++/VB6 function name is simply the function name with the preceding “Sx” removed. Explanations in this manual use VC++/VB6 function names. When programming in a .NET environment, remove the “Sx” from function names.

Asynchronous Messages Notification
Asynchronous messaging is accomplished through the “Event” method.
For details, see chapter 2.

Structures and Constants
Structure and constant definition is fundamentally the same as in VC++/VB6, but constants are defined using enum.
Since .NET structures and constants can be easily inferred from the structures and constants listed in this manual for VC++, they have been left out of the manual.

Using the Control Library (Reference)
1. In Visual Studio, click Tools then Choose Toolbox Items to open the Choose Toolbox Items dialog box. Click Browse in the .NET Framework Components tab.

The Open dialog box appears.
2. Select SxAPI.NET.dll, and then click Open.

3. Select the SxAPI check box, and then click OK. SxAPI appears in the Toolbox.

4. Drag the SxAPI icon from the Toolbox to the desired form.
2.1 Event Conditions

An application program receives an SxAPI event as a 32-bit integer with the conditions that triggered the event assigned to the integer’s bits. Application programs may receive multiple event conditions at the same time. If two events with the same condition are raised in brief succession, they may be consolidated into the same event. In other words, the number of times that an application program receives an event may be less than the number of times that the event was actually raised.

You can specify the conditions that you want to be notified of when you generate an event handler using SxCreateEvent().

The table below lists the conditions that can trigger events.

<table>
<thead>
<tr>
<th>Definition (.NET definitions are written in italics)</th>
<th>Bit Assignment</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>SX_EV_ACQ_START EV.ACQ_START</td>
<td>0x00010000</td>
<td>Measurement started.</td>
</tr>
<tr>
<td>SX_EV_ACQ_STOP EV.ACQ_STOP</td>
<td>0x00020000</td>
<td>Measurement stopped.</td>
</tr>
<tr>
<td>SX_EV_TRIG_START EV.TRIG_START</td>
<td>0x00040000</td>
<td>Trigger detected (in trigger mode).</td>
</tr>
<tr>
<td>SX_EV_TRIG_END EV.TRIG_END</td>
<td>0x00080000</td>
<td>Trigger measurement stopped (occurs after each acquisition).</td>
</tr>
<tr>
<td>SX_EV_ACQ_DATA_READY EV.ACQ_DATA_READY</td>
<td>0x08000000</td>
<td>Specified number of data points were acquired in free run mode (occurs after each acquisition).</td>
</tr>
<tr>
<td>SX_EV_SAVE_START EV.SAVE_SATRT</td>
<td>0x02000000</td>
<td>PC auto-recording operation started.</td>
</tr>
<tr>
<td>SX_EV_SAVE_END EV.SAVE_END</td>
<td>0x10000000</td>
<td>PC auto-recording operation ended.</td>
</tr>
<tr>
<td>SX_EV_CHANNEL_ALARM EV.CHANNEL_ALARM</td>
<td>0x20000000</td>
<td>Channel alarm occurred.</td>
</tr>
</tbody>
</table>
2.2 Programming Methods

Using SxEvent.ocx in Visual Basic 6.0

Open a project in VB. Click Project > Components, and then select SxEvent ActiveX Control Module from the menu and add it to the project. The icon indicated in the figure below will appear.

![SxEvent.ocx](image)

Double-click the SxEvent.ocx icon shown in the figure above and paste it into a form for receiving events. The form should look like the figure below.

![Form with SxEvent.ocx icon](image)

Double-click the SxEvent.ocx icon that you pasted into the form to create the event handler.

There is only one event handler type, SRQ1.

The following is an example of a program that saves the most recent measured data to a file if the event condition is SX_EV_TRIG_END (occurs when a triggered measurement ends).

```vba
Private Sub SxEvent1_SRQ1(ByVal handle As Long, ByVal pattern As Long)
    If pattern And SX_EV_TRIG_END Then ' Save measured data if the event ' indicates that a triggered measurement has finished.
        Err = SxSaveAcqData(handle, -1, "C:\Data\filename")
        ' The next trigger is enabled (for when the SL1000 is in Normal ' mode)
        Err = SxEnableNextTrig(handle)
    End If
End Sub
```
Using `PreTranslateMessage` in Visual C++

When the destination window for a message is not specified (NULL is specified) in the first parameter of `SxInit()`, the message will be sent to the main window. Main window messages can be processed by overriding `CWinApp::PreTranslateMessage`.

```cpp
BOOL WeApiTestApp::PreTranslateMessage(MSG* pMsg)
{
    return CWinApp::PreTranslateMessage(pMsg);
}
```

The sent message ID can be retrieved with `RegisterWindowMessage(SX_WM_EVENT)`.

`SX_WM_EVENT` is defined as follows in `SxAPI.h`.

```cpp
#define SX_WM_EVENT "WM_YOKOGAWA_TM_SX_EVENT"
```

Follow the example below to send an `SxAPI` event to the window that you want it to be handled by.

This example sends an event to `CMainFrame`.

```cpp
BOOL WeApiTestApp::PreTranslateMessage(MSG* pMsg)
{
    //TODO: Add your specialized code here and/or call the base class.
    UINT msg = pMsg->message;
    if (msg == RegisterWindowMessage(SX_WM_EVENT))
        return m_pMainWnd->SendMessage(msg, pMsg->wParam, pMsg->lParam);
    else
        return CWinApp::PreTranslateMessage(pMsg);
}
```

Define the event handler with the receiving class.

```cpp
const UINT  wm_SxAPI = RegisterWindowMessage(SX_WM_EVENT);
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //AFX_MSG_MAP(CSampleDlg)
    ON_WM_CREATE()
    ON_WM_CLOSE()
    //AFX_MSG_MAP
    ON_REGISTERED_MESSAGE(wm_SxAPI, OnSxEvent)
END_MESSAGE_MAP()
```

Write the event handler.

```cpp
LRESULT CMainFrame::OnSxEvent(WPARAM wp, LPARAM lp)
{
    SX_HNDL_UNIT hUnit = wp;  // WPARAM is a unit handle.
    ULONG pattern = lp;       // LPARAM is an event pattern.
    if(pattern & SX_EV_TRIG_END)
    {
        // Handling of the triggered measurement end event.
    }
    return TRUE;
}
```
2.2 Programming Methods

Using SxAPI.NET in Visual C#

In Visual Studio, drag the SxAPI control onto the form and open Properties. Click to display a list of events, and then double-click Event.

The following event handling method will appear in the code.

Write the event handling code inside of the above method. The following is an example of a program that saves the most recent measured data to a file if the event condition is SX_EV_TRIG_END (occurs when a triggered measurement ends).

```csharp
private void sxAPI1_Event(object sender, SxAPI.SxEventArgs e)
{
    int ret;
    if((e.pattern & (uint)EV.TRIG_END) != 0) // When there is a triggered measurement end event
    {
        // Save measured data
        ret = sxAPI1.SaveAcqData(e.unitHndl, -1, "C:\Data\filename");
        // The next trigger is enabled (for when the SL1000 is in Normal mode)
        ret = sxAPI1.EnableNextTrig(e.unitHndl);
    }
}
```
### 3.1 Functions

#### Initialization and Ending

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize</td>
<td>SxInit</td>
<td>3-6</td>
</tr>
<tr>
<td>Exit</td>
<td>SxExit</td>
<td>3-6</td>
</tr>
<tr>
<td>Re-search</td>
<td>SxReSearch</td>
<td>3-7</td>
</tr>
</tbody>
</table>

#### Retrieving Device Information

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get unit group information</td>
<td>SxGetGroupInfo</td>
<td>3-8</td>
</tr>
<tr>
<td>Get number of units</td>
<td>SxGetUnitNum</td>
<td>3-8</td>
</tr>
<tr>
<td>Get unit information</td>
<td>SxGetUnitInfo</td>
<td>3-9</td>
</tr>
<tr>
<td>Get module information</td>
<td>SxGetModInfo</td>
<td>3-9</td>
</tr>
</tbody>
</table>

#### Opening and Closing Handles

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open unit group</td>
<td>SxOpenGroup</td>
<td>3-10</td>
</tr>
<tr>
<td>Close unit group</td>
<td>SxCloseGroup</td>
<td>3-10</td>
</tr>
<tr>
<td>Open unit</td>
<td>SxOpenUnit</td>
<td>3-11</td>
</tr>
<tr>
<td>Close unit</td>
<td>SxCloseUnit</td>
<td>3-11</td>
</tr>
</tbody>
</table>

#### Retrieving Communication Handles

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get communication handle</td>
<td>SxMyCommHndl</td>
<td>3-12</td>
</tr>
<tr>
<td>Get unit group handle</td>
<td>SxMyGrpHndl</td>
<td>3-12</td>
</tr>
<tr>
<td>Get unit handle</td>
<td>SxMyUnitHndl</td>
<td>3-12</td>
</tr>
<tr>
<td>Get unit handle (by unit number)</td>
<td>SxUnitHndl</td>
<td>3-13</td>
</tr>
<tr>
<td>Get measuring group handle</td>
<td>SxMyMeasgrpHndl</td>
<td>3-13</td>
</tr>
<tr>
<td>Get measuring group handle (by measuring group number)</td>
<td>SxMeasgrpHndl</td>
<td>3-13</td>
</tr>
<tr>
<td>Get module handle</td>
<td>SxMyModHndl</td>
<td>3-14</td>
</tr>
<tr>
<td>Get module handle (by module number)</td>
<td>SxModHndl</td>
<td>3-14</td>
</tr>
<tr>
<td>Get channel handle</td>
<td>SxChHndl</td>
<td>3-14</td>
</tr>
</tbody>
</table>

#### Retrieving Values

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get channel number</td>
<td>SxChNo</td>
<td>3-15</td>
</tr>
<tr>
<td>Get module number</td>
<td>SxModNo</td>
<td>3-15</td>
</tr>
<tr>
<td>Get unit number</td>
<td>SxUnitNo</td>
<td>3-15</td>
</tr>
<tr>
<td>Get unit group number</td>
<td>SxGrpNo</td>
<td>3-15</td>
</tr>
<tr>
<td>Get measuring group number</td>
<td>SxMeasgrpNo</td>
<td>3-16</td>
</tr>
<tr>
<td>Get channel number</td>
<td>SxChNum</td>
<td>3-16</td>
</tr>
<tr>
<td>Get number of modules</td>
<td>SxModNum</td>
<td>3-16</td>
</tr>
<tr>
<td>Get unit number</td>
<td>SxUnitNum</td>
<td>3-16</td>
</tr>
</tbody>
</table>

#### Measuring Group Settings

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup measuring group</td>
<td>SxSetupMeasgrp</td>
<td>3-17</td>
</tr>
</tbody>
</table>
## 3.1 Functions

### Communication Command Controls

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send command</td>
<td>SxSetControl</td>
<td>3-18</td>
</tr>
<tr>
<td>Send binary data</td>
<td>SxSetControlBinary</td>
<td>3-20</td>
</tr>
<tr>
<td>Send and receive command</td>
<td>SxGetControl</td>
<td>3-20</td>
</tr>
<tr>
<td>Send and receive command (binary data reception)</td>
<td>SxGetControlBinary</td>
<td>3-21</td>
</tr>
<tr>
<td>Get received parameter (copy to the specified buffer)</td>
<td>SxGetParam</td>
<td>3-21</td>
</tr>
<tr>
<td>Get received parameter (get location)</td>
<td>SxGetParamPos</td>
<td>3-22</td>
</tr>
</tbody>
</table>

### Event Controls

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create and enable event handler</td>
<td>SxCreateEvent</td>
<td>3-23</td>
</tr>
<tr>
<td>Delete event handler</td>
<td>SxDeleteEvent</td>
<td>3-23</td>
</tr>
<tr>
<td>Enable events</td>
<td>SxEnableEvent</td>
<td>3-24</td>
</tr>
<tr>
<td>Enable next trigger and related event notification</td>
<td>SxEnableNextTrig</td>
<td>3-24</td>
</tr>
</tbody>
</table>

### Measurement Condition Settings and Queries

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch measurement on/off</td>
<td>SxSetAcqSwitch</td>
<td>3-25</td>
</tr>
<tr>
<td>Query measurement on/off</td>
<td>SxGetAcqSwitch</td>
<td>3-25</td>
</tr>
<tr>
<td>Switch auto recording on/off</td>
<td>SxSetRecSwitch</td>
<td>3-26</td>
</tr>
<tr>
<td>Query auto recording on/off</td>
<td>SxGetRecSwitch</td>
<td>3-26</td>
</tr>
<tr>
<td>Set measuring mode</td>
<td>SxSetAcqMode</td>
<td>3-26</td>
</tr>
<tr>
<td>Query measuring mode</td>
<td>SxGetAcqMode</td>
<td>3-26</td>
</tr>
<tr>
<td>Set sampling clock</td>
<td>SxSetClockSource</td>
<td>3-27</td>
</tr>
<tr>
<td>Query sampling clock</td>
<td>SxGetClockSource</td>
<td>3-27</td>
</tr>
<tr>
<td>Set sample rate</td>
<td>SxSetSamplingRate</td>
<td>3-27</td>
</tr>
<tr>
<td>Query sample rate</td>
<td>SxGetSamplingRate</td>
<td>3-27</td>
</tr>
<tr>
<td>Set sample interval</td>
<td>SxSetSamplingInterval</td>
<td>3-28</td>
</tr>
<tr>
<td>Query sample interval</td>
<td>SxGetSamplingInterval</td>
<td>3-28</td>
</tr>
<tr>
<td>Set sample points</td>
<td>SxSetAcqLength</td>
<td>3-28</td>
</tr>
<tr>
<td>Query sample points</td>
<td>SxGetAcqLength</td>
<td>3-28</td>
</tr>
<tr>
<td>Set sampling time</td>
<td>SxSetAcqSpan</td>
<td>3-29</td>
</tr>
<tr>
<td>Query sampling time</td>
<td>SxGetAcqSpan</td>
<td>3-29</td>
</tr>
<tr>
<td>Set trigger mode</td>
<td>SxSetTrigMode</td>
<td>3-29</td>
</tr>
<tr>
<td>Query trigger mode</td>
<td>SxGetTrigMode</td>
<td>3-29</td>
</tr>
<tr>
<td>Set pre-trigger position</td>
<td>SxSetTrigPos</td>
<td>3-30</td>
</tr>
<tr>
<td>Query pre-trigger position</td>
<td>SxGetTrigPos</td>
<td>3-30</td>
</tr>
<tr>
<td>Set pre-trigger points</td>
<td>SxSetPretrigLength</td>
<td>3-30</td>
</tr>
<tr>
<td>Query pre-trigger points</td>
<td>SxGetPretrigLength</td>
<td>3-30</td>
</tr>
<tr>
<td>Set pre-trigger time</td>
<td>SxSetPretrigTime</td>
<td>3-31</td>
</tr>
<tr>
<td>Query pre-trigger time</td>
<td>SxGetPretrigTime</td>
<td>3-31</td>
</tr>
<tr>
<td>Set trigger delay</td>
<td>SxSetTrigDelay</td>
<td>3-31</td>
</tr>
<tr>
<td>Query trigger delay</td>
<td>SxGetTrigDelay</td>
<td>3-31</td>
</tr>
<tr>
<td>Set trigger holdoff</td>
<td>SxSetTrigHoldOff</td>
<td>3-32</td>
</tr>
<tr>
<td>Query trigger holdoff</td>
<td>SxGetTrigHoldOff</td>
<td>3-32</td>
</tr>
<tr>
<td>Set trigger count</td>
<td>SxSetTrigCount</td>
<td>3-32</td>
</tr>
<tr>
<td>Query trigger count</td>
<td>SxGetTrigCount</td>
<td>3-32</td>
</tr>
<tr>
<td>Set channel label</td>
<td>SxSetChLabel</td>
<td>3-33</td>
</tr>
<tr>
<td>Query channel label</td>
<td>SxGetChLabel</td>
<td>3-33</td>
</tr>
<tr>
<td>Query acquisition data capacity</td>
<td>SxGetAcqCapacity</td>
<td>3-33</td>
</tr>
</tbody>
</table>
### 3.1 Functions

#### Auto Recording Condition Settings and Queries

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set auto recording destination</td>
<td>SxSetRecDest</td>
<td>3-34</td>
</tr>
<tr>
<td>Query auto recording destination</td>
<td>SxGetRecDest</td>
<td>3-34</td>
</tr>
<tr>
<td>Set recording start condition</td>
<td>SxSetRecStartCond</td>
<td>3-34</td>
</tr>
<tr>
<td>Query recording start condition</td>
<td>SxGetRecStartCond</td>
<td>3-35</td>
</tr>
<tr>
<td>Set recording start time</td>
<td>SxSetRecStartTime</td>
<td>3-35</td>
</tr>
<tr>
<td>Query recording start time</td>
<td>SxGetRecStartTime</td>
<td>3-35</td>
</tr>
<tr>
<td>Set recording stop condition</td>
<td>SxSetRecStopCond</td>
<td>3-36</td>
</tr>
<tr>
<td>Query recording stop condition</td>
<td>SxGetRecStopCond</td>
<td>3-36</td>
</tr>
<tr>
<td>Set recording end time</td>
<td>SxSetRecStopTime</td>
<td>3-37</td>
</tr>
<tr>
<td>Query recording end time</td>
<td>SxGetRecStopTime</td>
<td>3-37</td>
</tr>
<tr>
<td>Set recording time</td>
<td>SxSetRecSpan</td>
<td>3-37</td>
</tr>
<tr>
<td>Query recording time</td>
<td>SxGetRecSpan</td>
<td>3-37</td>
</tr>
<tr>
<td>Set number of points to record</td>
<td>SxSetRecExtClkPoints</td>
<td>3-38</td>
</tr>
<tr>
<td>Query number of points to record</td>
<td>SxGetRecExtClkPoints</td>
<td>3-38</td>
</tr>
<tr>
<td>Set recording interval mode</td>
<td>SxSetRecIntervalMode</td>
<td>3-38</td>
</tr>
<tr>
<td>Query recording interval mode</td>
<td>SxGetRecIntervalMode</td>
<td>3-38</td>
</tr>
<tr>
<td>Set recording interval</td>
<td>SxSetRecInterval</td>
<td>3-39</td>
</tr>
<tr>
<td>Query recording interval points</td>
<td>SxSetRecExtClkInterval</td>
<td>3-39</td>
</tr>
<tr>
<td>Query recording interval points</td>
<td>SxGetRecExtClkInterval</td>
<td>3-39</td>
</tr>
<tr>
<td>Set record count</td>
<td>SxSetRecTimes</td>
<td>3-40</td>
</tr>
<tr>
<td>Query record count</td>
<td>SxGetRecTimes</td>
<td>3-40</td>
</tr>
<tr>
<td>Set recording destination folder (PC)</td>
<td>SxSetRecFileFolder</td>
<td>3-40</td>
</tr>
<tr>
<td>Query recording destination folder (PC)</td>
<td>SxGetRecFileFolder</td>
<td>3-40</td>
</tr>
<tr>
<td>Set auto naming</td>
<td>SxSetRecFileAutoNaming</td>
<td>3-41</td>
</tr>
<tr>
<td>Query auto naming</td>
<td>SxGetRecFileAutoNaming</td>
<td>3-41</td>
</tr>
<tr>
<td>Set file name</td>
<td>SxSetRecFileName</td>
<td>3-41</td>
</tr>
<tr>
<td>Query file name</td>
<td>SxGetRecFileName</td>
<td>3-41</td>
</tr>
<tr>
<td>Set file order</td>
<td>SxSetRecFileOrder</td>
<td>3-42</td>
</tr>
<tr>
<td>Query file order</td>
<td>SxGetRecFileOrder</td>
<td>3-42</td>
</tr>
<tr>
<td>Set file count limit</td>
<td>SxSetRecCyclicFiles</td>
<td>3-42</td>
</tr>
<tr>
<td>Query file count limit</td>
<td>SxGetRecCyclicFiles</td>
<td>3-42</td>
</tr>
<tr>
<td>Set comment</td>
<td>SxSetRecFileComment</td>
<td>3-43</td>
</tr>
<tr>
<td>Query comment</td>
<td>SxGetRecFileComment</td>
<td>3-43</td>
</tr>
<tr>
<td>Get info. about file being recorded (PC)</td>
<td>SxGetRecCurrentFileInfo</td>
<td>3-43</td>
</tr>
<tr>
<td>Get info. about recorded files (PC)</td>
<td>SxGetRecLastFileInfo</td>
<td>3-43</td>
</tr>
</tbody>
</table>

#### Measurement Controls

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start measurement</td>
<td>SxAcqStart</td>
<td>3-44</td>
</tr>
<tr>
<td>Stop measurement</td>
<td>SxAcqStop</td>
<td>3-44</td>
</tr>
<tr>
<td>Execute latch (in Free Run mode)</td>
<td>SxAcqLatch</td>
<td>3-44</td>
</tr>
<tr>
<td></td>
<td>SxAcqLatchL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SxAcqLatchD</td>
<td></td>
</tr>
<tr>
<td>Confirm measure/save operation</td>
<td>SxIsRun</td>
<td>3-45</td>
</tr>
<tr>
<td>Execute manual trigger</td>
<td>SxExecManualTrig</td>
<td>3-45</td>
</tr>
</tbody>
</table>

#### Auto Recording Controls

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start recording</td>
<td>SxRecStart</td>
<td>3-46</td>
</tr>
<tr>
<td>Stop recording</td>
<td>SxRecStop</td>
<td>3-46</td>
</tr>
<tr>
<td>Divide file</td>
<td>SxRecDivide</td>
<td>3-46</td>
</tr>
</tbody>
</table>
## 3.1 Functions

### Acquisition and Deletion of Measured Data

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get acquisition data information (constants for converting physical values)</td>
<td>SxGetChannelInfo</td>
<td>3-47</td>
</tr>
<tr>
<td>Get latch interval sample points</td>
<td>SxGetLatchLength</td>
<td>3-47</td>
</tr>
<tr>
<td>Get latest acquisition number</td>
<td>SxGetLatestAcqNo</td>
<td>3-47</td>
</tr>
<tr>
<td>Get latest acquisition number (L)</td>
<td>SxGetLatestAcqNoL</td>
<td>3-47</td>
</tr>
<tr>
<td>Get latest acquisition number (D)</td>
<td>SxGetLatestAcqNoD</td>
<td>3-47</td>
</tr>
<tr>
<td>Get waveform data</td>
<td>SxGetAcqData</td>
<td>3-48</td>
</tr>
<tr>
<td>Get waveform data (L)</td>
<td>SxGetAcqDataL</td>
<td>3-48</td>
</tr>
<tr>
<td>Get waveform data (D)</td>
<td>SxGetAcqDataD</td>
<td>3-48</td>
</tr>
<tr>
<td>Get waveform data segment</td>
<td>SxGetAcqDataEx</td>
<td>3-49</td>
</tr>
<tr>
<td>Get waveform data segment (L)</td>
<td>SxGetAcqDataExL</td>
<td>3-49</td>
</tr>
<tr>
<td>Get waveform data segment (D)</td>
<td>SxGetAcqDataExD</td>
<td>3-49</td>
</tr>
<tr>
<td>Get data acquisition time</td>
<td>SxGetAcqTime</td>
<td>3-50</td>
</tr>
<tr>
<td>Get data acquisition time (L)</td>
<td>SxGetAcqTimeL</td>
<td>3-50</td>
</tr>
<tr>
<td>Get data acquisition time (D)</td>
<td>SxGetAcqTimeD</td>
<td>3-50</td>
</tr>
<tr>
<td>Get measuring group phase difference</td>
<td>SxGetAcqDelay</td>
<td>3-51</td>
</tr>
<tr>
<td>Get measuring group phase difference (L)</td>
<td>SxGetAcqDelayL</td>
<td>3-51</td>
</tr>
<tr>
<td>Get measuring group phase difference (D)</td>
<td>SxGetAcqDelayD</td>
<td>3-51</td>
</tr>
<tr>
<td>Create WDF file</td>
<td>SxSaveAcqData</td>
<td>3-51</td>
</tr>
<tr>
<td>Create WDF file (L)</td>
<td>SxSaveAcqDataL</td>
<td>3-51</td>
</tr>
<tr>
<td>Create WDF file (D)</td>
<td>SxSaveAcqDataD</td>
<td>3-51</td>
</tr>
<tr>
<td>Create WDF file (from waveform data segment)</td>
<td>SxSaveAcqDataEx</td>
<td>3-52</td>
</tr>
<tr>
<td>Create WDF file (from waveform data segment)</td>
<td>SxSaveAcqDataExL</td>
<td>3-52</td>
</tr>
<tr>
<td>Create WDF file (from waveform data segment)</td>
<td>SxSaveAcqDataExD</td>
<td>3-52</td>
</tr>
<tr>
<td>Get instantaneous values</td>
<td>SxGetCurrentData</td>
<td>3-53</td>
</tr>
<tr>
<td>Delete waveform data</td>
<td>SxClearAcqData</td>
<td>3-53</td>
</tr>
</tbody>
</table>

### Setup Data Access

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save setup data</td>
<td>SxSaveSetup</td>
<td>3-54</td>
</tr>
<tr>
<td>Load setup data</td>
<td>SxLoadSetup</td>
<td>3-54</td>
</tr>
<tr>
<td>Initialize setup data</td>
<td>SxInitSetup</td>
<td>3-54</td>
</tr>
</tbody>
</table>

### System-Related Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set unit group name</td>
<td>SxSetGroupName</td>
<td>3-55</td>
</tr>
<tr>
<td>Set unit name</td>
<td>SxSetUnitName</td>
<td>3-55</td>
</tr>
<tr>
<td>Execute calibration</td>
<td>SxExecCal</td>
<td>3-55</td>
</tr>
<tr>
<td>Execute self test</td>
<td>SxExecSelfTest</td>
<td>3-56</td>
</tr>
<tr>
<td>Get unit’s CPU temperature</td>
<td>SxGetCpuTemperature</td>
<td>3-56</td>
</tr>
<tr>
<td>Set unit’s clock</td>
<td>SxSetSystemClock</td>
<td>3-57</td>
</tr>
<tr>
<td>Query unit’s clock</td>
<td>SxGetSystemClock</td>
<td>3-57</td>
</tr>
<tr>
<td>Set DHCP</td>
<td>SxSetEthernetDHCP</td>
<td>3-58</td>
</tr>
<tr>
<td>Query DHCP</td>
<td>SxGetEthernetDHCP</td>
<td>3-58</td>
</tr>
<tr>
<td>Set IP address</td>
<td>SxSetEthernetIP</td>
<td>3-58</td>
</tr>
<tr>
<td>Query IP address</td>
<td>SxGetEthernetIP</td>
<td>3-58</td>
</tr>
<tr>
<td>Set subnet mask</td>
<td>SxSetEthernetNetMask</td>
<td>3-59</td>
</tr>
<tr>
<td>Query subnet address</td>
<td>SxGetEthernetNetMask</td>
<td>3-59</td>
</tr>
<tr>
<td>Set default gateway</td>
<td>SxSetEthernetGateway</td>
<td>3-59</td>
</tr>
<tr>
<td>Query default gateway</td>
<td>SxGetEthernetGateway</td>
<td>3-59</td>
</tr>
<tr>
<td>Set SNTP</td>
<td>SxSetSNTP</td>
<td>3-60</td>
</tr>
<tr>
<td>Query SNTP</td>
<td>SxGetSNTP</td>
<td>3-60</td>
</tr>
<tr>
<td>Set key lock</td>
<td>SxSetKeyLock</td>
<td>3-61</td>
</tr>
<tr>
<td>Query key lock</td>
<td>SxGetKeyLock</td>
<td>3-61</td>
</tr>
<tr>
<td>Get error code</td>
<td>SxGetError</td>
<td>3-61</td>
</tr>
</tbody>
</table>
### Internal Media Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number of drives</td>
<td>SxFileGetDriveNum</td>
<td>3-62</td>
</tr>
<tr>
<td>Query drive information</td>
<td>SxFileGetDriveInfo</td>
<td>3-62</td>
</tr>
<tr>
<td>Set current drive</td>
<td>SxFileSetCurrentDrive</td>
<td>3-62</td>
</tr>
<tr>
<td>Query current drive</td>
<td>SxFileGetCurrentDrive</td>
<td>3-63</td>
</tr>
<tr>
<td>Set current directory</td>
<td>SxFileChDir</td>
<td>3-63</td>
</tr>
<tr>
<td>Query current directory</td>
<td>SxFileCwDir</td>
<td>3-63</td>
</tr>
<tr>
<td>Create subdirectory</td>
<td>SxFileMkDir</td>
<td>3-64</td>
</tr>
<tr>
<td>Delete subdirectory</td>
<td>SxFileRmDir</td>
<td>3-64</td>
</tr>
<tr>
<td>Get number of files</td>
<td>SxFileGetFileNum</td>
<td>3-64</td>
</tr>
<tr>
<td>Get file information.</td>
<td>SxFileGetFileInfo</td>
<td>3-65</td>
</tr>
<tr>
<td>Delete file</td>
<td>SxFileDelete</td>
<td>3-65</td>
</tr>
<tr>
<td>Get file</td>
<td>SxFileGet SxFileGetM</td>
<td>3-66</td>
</tr>
<tr>
<td>Create file</td>
<td>SxFilePut SxFilePutM</td>
<td>3-66</td>
</tr>
<tr>
<td>Format internal hard disk</td>
<td>SxFormatHDD</td>
<td>6-67</td>
</tr>
</tbody>
</table>

### Debugging

<table>
<thead>
<tr>
<th>Function</th>
<th>Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set trace mode</td>
<td>SxTraceSetMode</td>
<td>3-68</td>
</tr>
<tr>
<td>Query trace mode</td>
<td>SxTraceGetMode</td>
<td>3-69</td>
</tr>
<tr>
<td>Output trace</td>
<td>SxTracePrint</td>
<td>3-69</td>
</tr>
<tr>
<td>Reset performance timer</td>
<td>SxResetTimer</td>
<td>3-69</td>
</tr>
<tr>
<td>Get performance timer</td>
<td>SxGetTimer</td>
<td>3-70</td>
</tr>
</tbody>
</table>
### 3.2 Initialization and Ending

**Initialization**

```c
int SxInit( HWND hWnd, int wire, char *option, SX_HNDL_COMM *hComm );
```

**Description**

Executing this function initializes the network, initiates the search for connected units, initializes the API runtime environment, and returns the communication handle of the specified network. When an application program uses SxAPI, it must execute this function first.

If an application executes this function, it must then execute SxExit() before it closes.

**Parameters**

- `hWnd`: Specifies the window to send a message to when an SRQ is received. If 0 is specified, the message will be sent to the main window. This parameter is invalid in the .NET interface.
- `wire`: Sets the type of network to connect to.
  - SX_WIRE_USB: USB
  - SX_WIRE_LAN: Ethernet
- `option`: Specifies an optional network text string. This parameter is invalid if the connection type is USB. If the connection type is Ethernet, this parameter specifies the subnet mask to use to search for units.
  - Example 1: "" (Null) All units within the subnet are searched.
  - Example 2: "255.255.255.0" Searches for units with the specified subnet mask.
  - Example 3: "192.168.21.3" Only searches for the unit with the specified address.
- `hComm`: Specifies where to store the initialized communication handle. If there is an error, 0 will be stored.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**Exit**

```c
int SxExit( SX_HNDL_COMM hComm );
```

**Description**

Executing this function closes the API runtime environment and the network drivers for the network specified by `hComm`. Always execute this function at the end of an application program.

**Parameters**

- `hComm`: A communication handle

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Re-Search

int SxReSearch( SX_HNDL_COMM hComm );
ERR ReSearch( HNDL hComm );

Description
Re-searches for units on the network specified by hComm. When this function is executed, previously retrieved handles with lower precedence than the communication handle (unit group handles, unit handles, module handles, channel handles, and measuring group handles) are released. So, when this function is executed, it is necessary to re-retrieve those low precedence handles. Be aware that the use of previously retrieved low-precedence handles will produce unexpected results.

Parameters
hComm A communication handle

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.3 Retrieving Device Information

Get Unit Group Information

```c
int SxGetGroupInfo( SX_HNDL_COMM | SX_HNDL_GROUP hAny, int groupNo, SX_INFO_GROUP *groupinfo );
```

```c
ERR GetGroupInfo( HNDL hAny, int groupNo, ref INFO_GROUP groupinfo);
```

**Description**

If hAny is a communication handle, this function retrieves the unit group information of the unit group specified by groupNo.

If hAny is a unit group handle, this function retrieves the unit group information of the group with that handle. In this case, the groupNo parameter is invalid.

**Parameters**

- **hAny** A communication handle or unit group handle
- **groupNo** A group number (0 to 15). This parameter is disabled when a unit group handle is specified for hAny.
- **groupinfo** Specifies where to store the unit group information.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Number of Units

```c
int SxGetUnitNum( SX_HNDL_COMM hComm, int groupNo, int *unitNum );
```

```c
ERR GetUnitNum( HNDL hComm, int groupNo, ref int unitNum );
```

**Description**

Retrieves the number of units in the unit group specified by groupNo.

**Parameters**

- **hComm** A communication handle
- **groupNo** A group number (0 to 15)
- **unitNum** Specifies where to store the unit number

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Get Unit Information

**Function Details**

```c
int SxGetUnitInfo( SX_HNDL_COMM | SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int groupNo, int unitNo, SX_INFO_UNIT *unitInfo);
ERR GetUnitInfo( HNDL hComm, int groupNo, int unitNo, ref INFO_UNIT unitInfo );
```

**Description**

If `hAny` is a communication handle, this function retrieves the unit information of the unit specified by `unitNo` that is in the unit group specified by `groupNo`.

If `hAny` is a unit group handle, `groupNo` is ignored and this function retrieves the unit information list of the unit by `unitNo`.

If `hAny` is a unit handle, this function retrieves the unit information list of the unit with that handle, and `groupNo` and `unitNo` are ignored.

An error is returned if there are two or more units that meet the conditions specified.

**Parameters**

- **hAny**: Any kind of handle
- **groupNo**: A group number (0 to 15)
- **unitNo**: A unit number (0 to 7)
- **unitInfo**: Specifies where to store the unit information

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetUnitInfoS( SX_HNDL_COMM hComm, int groupNo, int index, SX_INFO_UNIT *unitInfo );
ERR GetUnitInfoS( HNDL hComm, int groupNo, int index, ref INFO_UNIT unitInfo );
```

**Description**

Retrieves the unit information of the unit in the group specified by `groupNo` with the specified index (the order in which the units were found, starting with 0). The maximum number that can be set for the index is equal to the number of units acquired with `SxGetUnitNum()`–1.

**Parameters**

- **hComm**: A communication handle
- **groupNo**: A group number (0 to 15)
- **index**: An index number (from 0 to the number of units – 1)
- **unitInfo**: Specifies where to store the unit information

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Get Module Information

**Function Details**

```c
int SxGetModInfo( SX_HNDL_MOD hMod, SX_INFO_MOD *modInfo );
ERR GetModInfo( HNDL hMod, ref INFO_MOD modInfo );
```

**Description**

Retrieves the module information of the module specified by `hMod`.

**Parameters**

- **hMod**: A module handle
- **modInfo**: Specifies where to store the module information

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### 3.4 Opening and Closing Handles

#### Open Unit Group

```c
int SxOpenGroup( SX_HNDL_COMM hComm, int groupNo, SX_HNDL_GROUP *hGrp );
int SxOpenGroupEx( SX_HNDL_COMM hComm, int groupNo, SX_HNDL_GROUP *hGrp, int comm_tout, int alive_tout );
```

**Description**

Retrieves the handle of the unit group specified by `groupNo` with the communication handle specified by `hComm`. Returns an error if the specified unit group does not exist, or if it has an invalid ID.

You can specify the communication timeout and connection timeout with `SxOpenGroupEx()`.

If you use `SxOpenGroup()`, the communication timeout is set at 5 s, and the connection timeout is set at 10 s.

Be sure to close the unit group handles retrieved with these functions with `SxCloseGroup()` before closing the communication handle with `SxExit()`.

**Parameters**

- **hComm**: A communication handle
- **groupNo**: A unit group number (0 to 15)
- **hGrp**: Specifies where to store the unit group handle.
- **comm_tout**: Communication timeout in seconds.
- **alive_tout**: Connection timeout in seconds.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

#### Close Unit Group

```c
int SxCloseGroup( SX_HNDL_GROUP hGrp );
```

**Description**

Closes the unit group specified by `hGrp`.

Be sure to close the unit group handles retrieved with `SxOpenGroup()` or `SxOpenGroupEx()` with this function before closing the communication handle with `SxExit()`.

**Parameters**

- **hGrp**: A unit group handle

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### 3.4 Opening and Closing Handles

#### Open Unit

```c
int SxOpenUnit( SX_HNDL_COMM hComm, char *address, SX_HNDL_UNIT *hUnit );
int SxOpenUnitEx( SX_HNDL_COMM hComm, char *address, SX_HNDL_UNIT *hUnit, int comm_tout, int alive_tout );
```

**ERR OpenUnit( HNDL hComm, string address, ref HNDL hUnit );**

**ERR OpenUnitEx( HNDL hComm, string address, ref HNDL hUnit, int comm_tout, int alive_tout );**

**Description**

Retrieves the handle of a unit with the specified address in the network specified by `hComm`. An error is returned if the unit with the specified address does not exist.

You can specify the communication timeout and connection timeout with `SxOpenUnitEx()`. If you use `SxOpenUnit()`, the communication timeout is set at 5 s, and the connection timeout is set at 10 s.

To control measurement and auto recording, a unit group handle is necessary. Use a function such as `SxOpenGroup` to open a unit group.

Be sure to close the unit group handles acquired with these functions with `SxCloseUnit()` before closing the communication handle with `SxExit()`.

**Parameters**

- **hComm**
  A communication handle

- **address**
  An address

  - Use a serial number for USB (e.g., "12A456789")
  - Use an IP address for Ethernet (e.g., "192.168.21.3")

- **hUnit**
  Specifies where to store the unit handle.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

#### Close Unit

```c
int SxCloseUnit( SX_HNDL_UNIT hUnit );
```

**ERR CloseUnit( HNDL hUnit );**

**Description**

Closes the unit specified by `hUnit`.

Be sure to close the unit handles acquired with `SxOpenUnit()` or `SxOpenUnitEx()` with this function before closing the communication handle with `SxExit()`.

**Parameters**

- **hUnit**
  A unit handle

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.5 Handle Acquisition

Get Communication Handle
SX_HNDL_COMM SxMyCommHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny );
HNDL MyCommHndl( HNDL hAny );
Description
Returns the communication handle of the network that contains the object specified by hAny.
Parameters
hAny Any kind of handle
Return Value
If the function succeeds, the return value is a communication handle. If the function fails, the return value is zero.

Get Unit Group Handle
SX_HNDL_GROUP SxMyGrpHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny );
HNDL MyGrpHndl( HNDL hAny );
Description
Returns the unit group handle that contains the object specified by hAny.
If hAny is a unit group, the function returns hAny.
Parameters
hAny Any kind of handle
Return Value
If the function succeeds, the return value is a unit group handle. If the function fails, the return value is zero.

Get Unit Handle
SX_HNDL_UNIT SxMyUnitHndl(SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny );
HNDL MyUnitHndl( HNDL hAny );
Description
Returns the handle of the unit that contains the object specified by hAny.
If hAny is a unit, the function returns hAny.
Parameters
hAny Any kind of handle
Return Value
If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.
3.5 Handle Acquisition

Get Unit Handle (By Unit Number)

SX_HNDL_UNIT SxUnitHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int unitNo);
HNDL UnitHndl( HNDL hAny, int unitNo );

Description
Returns the handle of the unit specified by unitNo in the unit group that contains the object specified by hAny.
If hAny is a unit, the function returns hAny. (The unitNo parameter is invalid.)
If hAny is a module or channel, the function returns the handle of the unit that contains hAny. (The unitNo parameter is invalid.)

Parameters
- hAny Any kind of handle
- unitNo A unit number (0 to 7)

Return Value
If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.

Get Measuring Group Handle

SX_HNDL_MEASGRP SxMyMeasgrpHndl( SX_HNDL_MOD | SX_HNDL_CH hAny );
HNDL MyMeasgrpHndl( HNDL hAny );

Description
Returns the handle of the measuring group that contains the object specified by hAny.

Parameters
- hAny Any kind of handle

Return Value
If the function succeeds, the return value is a unit handle. If the function fails, the return value is zero.

Get Measuring Group Handle (By Measuring Group Number)

SX_HNDL_MEASGRP SxMeasgrpHndl( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int sampleNo );
HNDL MeasgrpHndl( HNDL hAny, int sampleNo );

Description
Returns the handle of the measuring group specified by sampleNo in the unit group that contains the object specified by hAny.
If hAny is a measuring group, the function returns hAny. (The sampleNo parameter is invalid.)
If hAny is a module or channel, the function returns the measuring group that contains hAny. (The sampleNo parameter is invalid.)

Parameters
- hAny Any kind of handle
- sampleNo A measuring group number (0 to 3)

Return Value
If the function succeeds, the return value is a measuring group handle. If the function fails, the return value is zero.
3.5 Handle Acquisition

Get Module Handle
SX_HNDL_MOD SxMyModHndl(SX_HNDL_MOD | SX_HNDL_CH hAny);
HNDL MyModHndl( HNDL hAny );

Description
If hAny is a channel, the function returns the handle of the module that contains hAny.
If hAny is a module, the function returns hAny.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a module handle. If the function fails, the
return value is zero.

Get Module Handle (By Module Number)
SX_HNDL_MOD SxModHndl( SX_HNDL_GROUP | SX_HNDL_UNIT hAny | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int moduleNo);
HNDL ModHndl( HNDL hAny, int moduleNo );

Description
Returns the handle of the module specified by moduleNo in the unit that contains
the object specified by hAny. If hAny is a unit group or measuring group, specify the
moduleNo with a sequence number starting with unit 0.
If hAny is a module, the function returns hAny. (The moduleNo parameter is invalid.)
If hAny is a channel, the function returns the handle of the module that contains hAny. (The
moduleNo parameter is invalid).

Parameters
hAny Any kind of handle
moduleNo A module number (if hAny is SX_HNDL_GROUP, use a sequence number
starting with unit 0.)

Return Value
If the function succeeds, the return value is a module handle. If the function fails, the
return value is zero.

Get Channel Handle
SX_HNDL_CH SxChHndl(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int channelNo);
HNDL ChHndl( HNDL hAny, int channelNo );

Description
Retrieves the handle of the channel specified by channelNo that belongs to the object
specified by hAny.
If hAny is a unit group, specify the channel with a sequence number starting with unit 0.
If hAny is a unit, specify the channel with the unit’s logic channel number.
If hAny is a module, specify the channel with the module’s channel number, based on its
order with the first channel being 0.
If hAny is a channel, the function returns hAny. (The channelNo parameter is invalid.)
If hAny is a measuring group, specify the channel with a sequence number starting with
unit 0 (leaving out channels in other measuring groups).

Parameters
hAny Any kind of handle
channelNo A channel number

Return Value
If the function succeeds, the return value is a channel handle. If the function fails, the
return value is zero.
3.6 Value Retrieval

Get Channel Number
int SxChNo(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny );
int ChNo( HNDL hAny );

Description
Returns the logic channel number of the first channel in the object specified by hAny.
The function returns 0 if hAny is a unit group.
If hAny is a unit, the function returns the logic channel number of the first channel in that unit.
If hAny is a module, the function returns the logic channel number of the first channel in that module.
If hAny is a channel, the function returns the logic channel number of the channel.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a logic channel number. If the function fails, the return value is –1.

Get Module Number
int SxModNo( SX_HNDL_MOD | SX_HNDL_CH hAny );
int ModNo( HNDL hAny );

Description
Returns the slot number of the module that contains the object specified by hAny.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a slot number. If the function fails, the return value is –1.

Get Unit Number
int SxUnitNo( SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny );
int UnitNo( HNDL hAny );

Description
Returns the unit number of the unit that contains the object specified by hAny.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a unit number. If the function fails, the return value is –1.

Get Unit Group Number
int SxGrpNo(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);
int GrpNo( HNDL hAny );

Description
Returns the unit group number of the group that contains the object specified by hAny.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a unit number. If the function fails, the return value is –1.
3.6 Value Retrieval

Get Measuring Group Number
int SxMeasgrpNo( SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny);
int MeasgrpNo( HNDL hAny );

Description
Returns the measuring group number of the measuring group that contains the object
specified by hAny.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is a measuring group number. If the function
fails, the return value is –1.

Get Channel Number
int SxChNum(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH |
SX_HNDL_MEASGRP hAny);
int ChNum( HNDL hAny );

Description
Returns the total number of channels in the object specified by hAny.
If hAny is a channel, the function returns 1.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is the number of channels. If the function fails,
the return value is –1.

Get Module Number
int SxModNum( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD |
SX_HNDL_MEASGRP hAny );
int ModNum( HNDL hAny );

Description
Returns the total number of modules in the object specified by hAny.
If hAny is a module, the function returns 1.
If hAny is a measuring group, the function returns the total number of modules in the
group.

Parameters
hAny Any kind of handle

Return Value
If the function succeeds, the return value is the number of modules. If the function fails,
the return value is –1.

Get Unit Number
int SxUnitNum( SX_HNDL_GROUP hGrp );
int UnitNum( HNDL hGrp );

Description
Returns the number of units in the unit group specified by hGrp.

Parameters
hGrp A unit group handle

Return Value
If the function succeeds, the return value is the number of units. If the function fails, the
return value is –1.
3.7 Measuring Group Settings

Setup Measuring Group

```c
int SxSetupMeasgrp( SX_HNDL_MOD hMod, int measgrpNo);
ERR SetupMeasgrp( HNDL hMod, int measgrpNo );
```

**Description**
Assigns the module specified by hMod to the measuring group specified by measgrpNo.

**Parameters**
- `hMod` A module handle
- `measgrpNo` A measuring group number (0 to 3)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.8 Communication Command Controls

Send Command

Send a program message to the unit specified by hAny, waits for processing to finish, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands at once, separate each command with a semicolon.

Even if the program message contains a query (a command that ends with a question mark), the response to the command will be discarded.

For information about what commands can be sent, see chapter 4.

If \(<ch>\) or \(<mo>\) is contained in the command string, the actual string that will be sent will follow the pattern of the examples shown below.

When the Command String Contains \(<ch>\)

If hAny is a unit group, the command will be copied for as many times as there are channels in the unit group, and \(<ch>\) will be replaced by the physical channel number.

Example:

Command string: \(\text{"CHAN}<ch>\text{:RANG 10.0"}\)

String sent to unit 0:
\(\text{"CHAN1:RANG 10.0}; \text{CHAN2:RANG 10.0}; \ldots ; \text{CHAN16:RANG 10.0"}\)

String sent to unit 1:
\(\text{"CHAN17:RANG 10.0}; \text{CHAN18:RANG 10.0}; \ldots ; \text{CHAN32:RANG 10.0"}\)

If hAny is a unit, the command will be copied for as many times as there are channels in the unit, and \(<ch>\) will be replaced by the physical channel number.

Example:

Command string: \(\text{"CHAN}<ch>\text{:RANG 10.0"}\)

String sent to the unit:
\(\text{"CHAN17:RANG 10.0}; \text{CHAN18:RANG 10.0}; \ldots ; \text{CHAN32:RANG 10.0"}\)

If hAny is a module, the command will be copied for as many times as there are channels in the module, and \(<ch>\) will be replaced by the physical channel number.

Example:

Command string: \(\text{"CHAN}<ch>\text{:RANG 10.0"}\)

String sent to the unit:
\(\text{"CHAN5:RANG 10.0}; \text{CHAN6 10.0"}\)

If hAny is a channel, \(<ch>\) will be replaced by the channel's physical channel number.

Example:

Command string: \(\text{"CHAN}<ch>\text{:RANG 10.0"}\)

String sent to the unit:
\(\text{"CHAN11:RANG 10.0"}\)
If \( h\text{Any} \) is a measuring group, the command will be copied for as many times as there are channels in the measuring group, and \(<\text{ch}>\) will be replaced by the physical channel number.

Example:
- Command string: \"CHAN<\text{ch}>:RANG 10.0\"
- String sent to unit 0: \"CHAN3:RANG 10.0;CHAN4:RANG 10.0;\"
- String sent to unit 1: \"CHAN19:RANG 10.0;CHAN20:RANG 10.0;\"

When the Command String Contains \(<\text{mo}>\)

If \( h\text{Any} \) is a unit group, the command will be copied for as many times as there are modules in the unit group, and \(<\text{mo}>\) will be replaced by the physical slot number.

Example:
- Command string: \":TIM:MODU<\text{mo}>:GROU 4\"
- String sent to unit 0: \":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4;\"
- String sent to unit 1: \":TIM:MODU9:GROU 4; ... ;:TIM:MODU15:GROU 4;:TIM:MODU16:GROU 4;\"

If \( h\text{Any} \) is a unit, the command will be copied for as many times as there are modules in the unit, and \(<\text{mo}>\) will be replaced by the physical slot number.

Example:
- Command string: \":TIM:MODU<\text{mo}>:GROU 4\"
- String sent to the unit: \":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4;\"

If \( h\text{Any} \) is a module, \(<\text{mo}>\) will be replaced by the module's physical slot number.

Example:
- Command string: \":TIM:MODU<\text{mo}>:GROU 4\"
- String sent to the unit: \":TIM:MODU3:GROU 4;\"

If \( h\text{Any} \) is a channel, \(<\text{mo}>\) will be replaced by the physical slot number of the module that contains the channel.

Example:
- Command string: \":TIM:MODU<\text{mo}>:GROU 4\"
- String sent to the unit: \":TIM:MODU3:GROU 4;\"

If \( h\text{Any} \) is a measuring group, the command will be copied for as many times as there are modules in the measuring group, and \(<\text{mo}>\) will be replaced by the physical slot number.

Example:
- Command string: \":TIM:MODU<\text{mo}>:GROU 4\"
- String sent to unit 0: \":TIM:MODU1:GROU 4;:TIM:MODU2:GROU 4; ... ;:TIM:MODU8:GROU 4;\"
- String sent to unit 1: \":TIM:MODU9:GROU 4; ... ;:TIM:MODU15:GROU 4;:TIM:MODU16:GROU 4;\"

Parameters
- \( h\text{Any} \): Any kind of handle
- \( \text{msg} \): The initial address of the program message to be sent

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Send Binary Data

int SxSetControlBinary( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int len );

ERR SetControlBinary( HNDL hAny, string msg, any[] buf, int len );
ERR SetControlBinary( HNDL hAny, string msg, any[] buf );

Description
Sends a program message with binary parameters to the unit specified by hAny, waits for processing to finish, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. This function cannot send multiple commands. Also, the function will not receive responses to queries (commands that end with question marks).

For information about what commands can be sent, see chapter 4.

If the command string contains `<ch>` or `<mo>`, it will be copied and altered in the same ways as with SxSetControl().

Parameters
- **hAny** Any kind of handle
- **msg** The initial address of the header of the program message to be sent
- **buf** The initial address of the binary parameters to be sent
- **len** The length of the parameters to be sent, specified in bytes

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Send and Receive Command

int SxGetControl( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int blen, int *rlen );

ERR GetControl( HNDL hAny, string req, ref string rep, int blen, ref int rlen );
ERR GetControl( HNDL hAny, string req, ref string rep, int blen );
ERR GetControl( HNDL hAny, string req, ref string rep );

Description
Sends a program message to the unit specified by hAny, receives a response, and then exits from the function.

If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands, separate each command with a semicolon.

If there are no queries (commands that end with question marks) in the program message, a timeout error will result.

For information about what commands can be sent, see chapter 4.

If the command string contains `<ch>` or `<mo>`, it will be copied and altered in the same ways as with SxSetControl().

Parameters
- **hAny** Any kind of handle
- **msg** The initial address of the program message to be sent
- **buf** Where to store the response message
- **blen** The size of the area for storing the response message, specified in bytes
- **rlen** Where to store the length (in bytes) of the response message string

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Send and Receive Command (Binary Data Reception)

```c
int SxGetControlBinary(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, char *msg, char *buf, int blen, int *rlen);
```

**Description**
Sends a program message to the unit specified by hAny, receives a response, and then exits from the function. The received response message’s parameters must be in binary format.

- If hAny is a unit group or measuring group, the message will be sent to all units in the group. To send multiple commands, separate each command with a semicolon. You cannot send multiple queries.
- If there are no queries (commands that end with question marks) in the program message, a timeout error will result.
- For information about what commands can be sent, see chapter 4.
- If the command string contains `<ch>` or `<mo>,” it will be copied and altered in the same ways as with SxSetControl().

**Parameters**
- **hAny**  Any kind of handle
- **msg**  The initial address of the program message to be sent
- **buf**  Where to store the response message (that contains binary parameters)
- **blen**  The size of the area for storing the response message, specified in bytes
- **rlen**  Where to store the length of the response message, specified in bytes

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Received Parameters (Copy to Specified Buffer)

```c
int SxGetParam(char *rep, int pos, int idx, char *buf, int blen);
int SxGetParamStr(char *rep, int pos, int idx, char *buf, int blen);
ERR GetParam(string rep, int pos, int idx, ref string buf, int blen);
ERR GetParamStr(string rep, int pos, int idx, ref string buf);
```

**Description**
Extracts the parameter string with the specified number from the received message string and stores it in buf.

- The SxGetParamStr() function removes the double quotation marks from the beginning and end of a parameter string that starts with a double quotation mark before storing it.
- If the parameter with the specified number does not exist, the function will return a nonzero error code.

**Parameters**
- **rep**  The initial address of the received message string.
- **pos**  The command number (starting with zero). Commands are separated with a semicolon.
- **idx**  The element number (starting with zero). Elements are pieces of text separated with commas.
- **buf**  Where to store the extracted parameter string
- **blen**  The size of the area for storing the extracted parameter string.

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Get Received Parameters (Get Location)

char* SxGetParamPos( char *rep, int pos, int idx );

**Description**

Returns the location of the parameter string with the specified number in the received
message string.

This function can only be used with VC++.

**Parameters**

- **rep**: The initial address of the received message string.
- **pos**: The command number (starting with zero).
- **idx**: The element number (starting with zero). Elements are pieces of text
  separated with commas.

**Return Value**

The function returns the initial address of the specified parameter text string. If there is
an error, the function returns NULL.
3.9 Event Controls

Create and Enable Event Handler

```c
int SxCreateEvent( SX_HNDL_GROUP hGrp, HWND hWnd, ULONG enable );
```

**Description**

Allows a message to be sent when the unit group specified by hGrp creates an event.

- Enables events specified with a 1 bit in the enable parameter.
- hWnd specifies the handle of the window to which the message will be sent.
- hWnd is invalid in the .NET interface.
- If hWnd is NULL, the message will be sent to the main window.
- The window message name is "WM_YOKOGAWA_TM_SX_EVENT."
- The window message ID can be acquired with this function:
  `RegisterWindowMessage("WM_YOKOGAWA_TM_SX_EVENT").`
- The window message’s WPARAM is the event source’s unit handle. LPARAM is the event’s bit pattern.
- Be sure to close the event handlers created with this function with SxDeleteEvent() before closing the communication handle with SxExit().
- For details about event handling, see chapter 2.

**Parameters**

- **hGrp** A unit group handle
- **hWnd** The handle of the window to which the message will be sent (If the value is NULL, the message will be sent to the main window.)
- **enable** The event enabling bit pattern (enable with 1, disable with 0)

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Event Handler

```c
int SxDeleteEvent( SX_HNDL_GROUP hGrp );
```

**Description**

Closes the event handler of the unit group specified by hGrp.

- Be sure to use this function to close the event handlers created with SxCreateEvent() before closing the communication handle with SxExit().

**Parameters**

- **SX_HNDL_GROUP hGrp** A unit group handle

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Enable Events

**int SxEnableEvent( SX_HNDL_GROUP hGrp, ULONG enable );**

**ERR EnableEvent( HNDL hGrp, ULONG enable );**

**Description**

Enables the bits that allow events to occur for the unit group specified by hGrp. Bits set to 1 are enabled, and bits set to 0 are disabled. Use this function to change the enable pattern specified by the enable parameter of SxCreateEvent().

**Parameters**

- **hGrp**: A unit group handle
- **enable**: Bit pattern to enable events (1 to enable, 0 to mask)

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Enable Next Trigger and Related Event Notification

**int SxEnableNextTrig( SX_HNDL_GROUP hGrp );**

**ERR EnableNextTrig(HNDL hGrp);**

**Description**

Enables the next trigger detection and the generation of the TRG and TRIG_END events for the unit group specified by hGrp.

So that units do not produce communication and PC processing problems by sending multiple TRG_START and TRIG_END events in Single (N) mode, events after the first event will not be raised until this function is executed. To receive TRG_START and TRIG_END events after the first event, this function must be executed after each TRG_START or TRIG_END event is received to allow the raising of the next event.

In Normal mode, the SL1000 begins detection of the next trigger after receiving permission from the PC. So it is necessary to use this function to enable trigger detection in order to enable triggering after the first trigger.

**Parameters**

- **hGrp**: A unit group handle

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.10 Measurement Condition Settings and Queries

Query or Set Measurement On/Off

int SxSetAcqSwitch(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqSW);

Description
Turns the measurement of the channels contained in the object specified by hAny on or off.

Parameters
hAny Any kind of handle
acqSW 0: Off
       1: On

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetAcqSwitch( SX_HNDL_CH hCh, int *acqSW );

ERR GetAcqSwitch( HNDL hCh, ref BOOL acqSW );

Description
Queries the on/off status of the measurement of the channel specified by hCh.

Parameters
hCh A channel handle
acqSW Where to store the measurement on/off information
       0: Off
       1: On

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Recording On/Off

```c
int SxSetRecSwitch( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int recSW );
```

**ERR** SetRecSwitch( HNDL hAny, BOOL recSW );

**Description**

Turns the recording of the channels contained in the object specified by hAny on or off.

**Parameters**

- **hAny**: Any kind of handle
- **recSW**: 0: Off
  1: On

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecSwitch( SX_HNDL_CH hCh, int *recSW );
```

**ERR** GetRecSwitch( HNDL hCh, ref BOOL recSW );

**Description**

Queries the on/off status of the recording of the channel specified by hCh.

**Parameters**

- **hCh**: A channel handle
- **recSW**: Where to store the recording on/off information
  0: Off
  1: On

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Measuring Mode

```c
int SxSetAcqMode( SX_HNDL_GROUP hGrp, int acqMode );
```

**ERR** SetAcqMode( HNDL hGrp, ACQMODE acqMode );

**Description**

Sets the measuring mode of the unit group specified by hGrp to the mode specified by acqMode.

The acquisition mode is fixed at Normal. You cannot select the Envelope or Box Average modes.

**Parameters**

- **hGrp**: A unit group handle
- **acqMode**: SX_ACQ_FREE: Free run mode
  SX_ACQ_TRIG: Trigger mode

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetAcqMode(SX_HNDL_GROUP hGrp, int *acqMode );
```

**ERR** GetAcqMode( HNDL hGrp, ref ACQMODE acqMode );

**Description**

Queries the measuring mode of the measuring group specified by hGrp.

**Parameters**

- **hGrp**: A unit group handle
- **acqMode**: Where to store the measuring mode information
  SX_ACQ_FREE: Free run mode
  SX_ACQ_TRIG: Trigger mode

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.10 Measurement Condition Settings and Queries

Query or Set Sampling Clock
int SxSetClockSource(SX_HNDL_GROUP hGrp, int clockSource);
**ERR SetClockSource( HNDL hGrp, CLOCKSOURCE clockSource );**

**Description**
Sets the source of the sampling clock of the unit group specified by hGrp to the source specified by clock.

**Parameters**
- hGrp: A unit group handle
- clockSource: SX_CLK_INT: The internal clock
  SX_CLK_EXT: An external clock

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetClockSource(SX_HNDL_GROUP hGrp, int *clockSource);
**ERR GetClockSource( HNDL hAny, ref CLOCKSOURCE clockSource );**

**Description**
Queries the source of the sampling clock of the unit group specified by hGrp.

**Parameters**
- hGrp: A unit group handle
- clockSource: Where to store the clock source information
  SX_CLK_INT: The internal clock
  SX_CLK_EXT: An external clock

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sample Rate
int SxSetSamplingRate(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double smplRate);
**ERR SetSamplingRate( HNDL hAny, double smplRate );**

**Description**
Sets the sample rate of the measuring group specified by hAny to the sample rate specified by smplRate.
If hAny is a unit or unit group, all of the measuring groups that hAny contains will be set to the same sample rate.

**Parameters**
- hAny: Any kind of handle
- smplRate: The sample rate frequency (in Hz)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetSamplingRate(SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double *smplRate);
**ERR GetSamplingRate( HNDL hAny, ref double smplRate );**

**Description**
Queries the sample rate of the measuring group specified by hAny.

**Parameters**
- hAny: Any kind of handle
- smplRate: Where to store the sample rate frequency (in Hz)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.10 Measurement Condition Settings and Queries

Query or Set Sample Interval

```c
int SxSetSamplingInterval( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double smplInterval);

ERR SetSamplingInterval( HNDL hAny, double smplInterval );
```

**Description**
Changes the sample rate of the measuring group specified by hAny to the inverse of the sample interval specified by smplInterval.
If hAny is a unit or unit group, all of the measuring groups that hAny contains will be set to the same sample rate.
The smplRate and smplInterval information for the measuring groups whose values are changed will be updated simultaneously.

**Parameters**
- `hAny`: Any kind of handle
- `smplRate`: The sample interval (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetSamplingInterval( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MEASGRP hAny, double *smplInterval);

ERR GetSamplingInterval( HNDL hAny, ref double smplInterval );
```

**Description**
Queries the inverse of the sample rate of the measuring group specified by hAny.

**Parameters**
- `hAny`: Any kind of handle
- `smplInterval`: Where to store the sample interval information (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Sample Points

```c
int SxSetAcqLength( SX_HNDL_GROUP hGrp, int len);

ERR SetAcqLength( HNDL hGrp, int len );
```

**Description**
Sets the number of measurement points of the unit group specified by hGrp to the length specified by len. Specify the number of measurement points with the number of measurement group 1 measurement points.

**Parameters**
- `hGrp`: A unit group handle
- `len`: The number of measurement points

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetAcqLength( SX_HNDL_GROUP hGrp, int *len );

ERR GetAcqLength( HNDL hGrp, ref int len );
```

**Description**
Queries thenumber of measurement points of the unit group specified by hGrp. The number of measurement points is specified with the number of measurement group 1 measurement points.

**Parameters**
- `hGrp`: A unit group handle
- `len`: Where to store the number of measurement points

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Sampling Time

```c
int SxSetAcqSpan( SX_HNDL_GROUP hGrp, double second);
```

**Description**
Sets the measuring time of the unit group specified by hGrp to the number of seconds specified by the second parameter.

This setting is invalid if an external sampling clock is being used.

**Parameters**
- `hGrp`: A unit group handle
- `second`: The measuring time in seconds

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetAcqSpan( SX_HNDL_GROUP hGrp, double *second );
```

**Description**
Queries the measuring time (in seconds) of the unit group specified by hGrp. The measuring time is specified with the number of measurement group 1 measuring time.

**Parameters**
- `hGrp`: A unit group handle
- `second`: The measuring time in seconds

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Trigger Mode

```c
int SxSetTrigMode( SX_HNDL_GROUP hGrp, int trigMode );
```

**Description**
Sets the trigger mode of the measuring group specified by hGrp to the mode specified by trigMode.

**Parameters**
- `hGrp`: A unit group handle
- `trigMode`: SX_TRIG_NORMAL: Normal mode
  - SX_TRIG_SINGLE: Single mode
  - SX_TRIG_NSINGLE: Single (N) mode

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetTrigMode( SX_HNDL_GROUP hGrp, int *trigMode );
```

**Description**
Queries the trigger mode of the measuring group specified by hGrp.

**Parameters**
- `hGrp`: A unit group handle
- `trigMode`: Where to store the trigger mode
  - SX_TRIG_NORMAL: Normal mode
  - SX_TRIG_SINGLE: Single mode
  - SX_TRIG_NSINGLE: Single (N) mode

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Query or Set Pre-Trigger Position

**int SxSetTrigPos( SX_HNDL_GROUP hGrp, double percent );**

**ERR SetTrigPos( HNDL hGrp, double percent );**

**Description**
Sets the pre-trigger length of the unit group specified by hGrp to a percentage (specified by percent) of the total measuring time.
This setting is invalid if an external sampling clock is being used.

**Parameters**
- hGrp: A unit group handle
- percent: The pre-trigger length (as a percentage)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Query or Set Pre-Trigger Points

**int SxSetPretrigLength( SX_HNDL_GROUP hGrp, int points );**

**ERR SetPretrigLength( HNDL hGrp, int points );**

**Description**
Sets the pre-trigger length of the unit group specified by hGrp to the specified number of measurement points. The length is specified with the number of measurement group 1 points that are equivalent to that length.

**Parameters**
- hGrp: A unit group handle
- points: The pre-trigger length (number of sample points)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Query or Set Pre-Trigger Points

**int SxGetTrigPos( SX_HNDL_GROUP hGrp, double *percent );**

**ERR GetTrigPos( HNDL hGrp, ref double percent );**

**Description**
Queries the pre-trigger length as a percentage of the total measuring time of the unit group specified by hGrp.

**Parameters**
- hGrp: A unit group handle
- percent: Where to store the pre-trigger length (as a percentage)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**int SxGetPretrigLength( SX_HNDL_GROUP hGrp, int *points );**

**ERR GetPretrigLength( HNDL hGrp, ref int points );**

**Description**
Queries the pre-trigger length in sample points of unit group specified by hGrp (pre-trigger length based on measuring group 1).

**Parameters**
- hGrp: A unit group handle
- points: Where to store the pre-trigger length (number of sample points)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.10 Measurement Condition Settings and Queries

Query or Set Pre-Trigger Time

```c
int SxSetPretrigTime( SX_HNDL_GROUP hGrp, double second );
```

```c
ERR SetPretrigTime( HNDL hGrp, double second );
```

**Description**
Sets the pre-trigger time of the unit group specified by hGrp to the number of seconds specified by second.
This function is invalid when the sample clock is set to an external signal.

**Parameters**
- **hGrp**: A unit group handle
- **second**: Pre-trigger time (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetPretrigTime( SX_HNDL_GROUP hGrp, double *second );
```

```c
ERR GetPretrigTime( HNDL hGrp, ref double second );
```

**Description**
Queries the pre-trigger time of the unit group specified by hGrp.
This function is invalid when the sample clock is set to an external signal.

**Parameters**
- **hGrp**: A unit group handle
- **second**: Where to store the pre-trigger time (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Trigger Delay

```c
int SxSetTrigDelay( SX_HNDL_GROUP hGrp, double second );
```

```c
ERR SetTrigDelay( HNDL hGrp, double second );
```

**Description**
Sets the trigger delay of the unit group specified by hGrp to the number of seconds specified by second.
This function is invalid when the sample clock is set to an external signal.

**Parameters**
- **hGrp**: A unit group handle
- **second**: Trigger delay (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetTrigDelay( SX_HNDL_GROUP hGrp, double *second );
```

```c
ERR GetTrigDelay( HNDL hGrp, ref double second );
```

**Description**
Queries the trigger delay of the unit group specified by hGrp.
This function is invalid when the sample clock is set to an external signal.

**Parameters**
- **hGrp**: A unit group handle
- **second**: Where to store the trigger delay (in seconds)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Trigger Holdoff

int SxSetTrigHoldOff( SX_HNDL_GROUP hGrp, double second );
ERR SetTrigHoldOff( HNDL hGrp, double second );

Description
Sets the trigger holdoff of the unit group specified by hGrp to the number of seconds specified by second.
This function is invalid when the sample clock is set to an external signal.

Parameters
- hGrp: A unit group handle
- second: Trigger holdoff (in seconds)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetTrigHoldOff( SX_HNDL_GROUP hGrp, double *second );
ERR GetTrigHoldOff( HNDL hGrp, ref double second );

Description
Queries the trigger holdoff of the unit group specified by hGrp.
This function is invalid when the sample clock is set to an external signal.

Parameters
- hGrp: A unit group handle
- second: Where to store the trigger holdoff (in seconds)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Trigger Count

int SxSetTrigCount( SX_HNDL_GROUP hGrp, int trigCount );
ERR SetTrigCount( HNDL hGrp, int trigCount );

Description
Sets the trigger count of the unit group specified by hGrp to the count specified by trigCount.
This function is invalid when the trigger mode is SX_TRIG_SINGLE.

Parameters
- hGrp: A unit group handle
- trigCount: The trigger count (0 indicates no limit)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetTrigCount( SX_HNDL_GROUP hGrp, int *trigCount );
ERR GetTrigCount( HNDL hGrp, ref int trigCount );

Description
Queries the trigger count of the unit group specified by hGrp.

Parameters
- hGrp: A unit group handle
- trigCount: Where to store the trigger count (0 indicates no limit)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.10 Measurement Condition Settings and Queries

Query or Set Channel Label

int SxSetChLabel( SX_HNDL_CH hCh, char *label );

ERR SetChLabel( HNDL hCh, string label );

Description
Sets the label of the channel specified by hCh to the string specified by the label parameter. Trying to set the label to a string longer than 7 characters will result in an error.

Parameters
- hCh: A channel handle
- label: The label character string (up to 8 characters including the null character)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetChLabel( SX_HNDL_CH hCh, char *label );

ERR GetChLabel( HNDL hCh, ref string label );

Description
Queries the label of the channel specified by hCh.

Parameters
- hCh: A channel handle
- label: Where to store the label string (8 bytes or more)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query Acquisition Data Capacity

int SxGetAcqCapacity( SX_HNDL_GROUP hGrp, int *size );

ERR GetAcqCapacity( HNDL hGrq, ref int size );

Description
Queries how much acquisition data can be stored in each channel in the unit group specified by hGrp. In Triggered mode, the acquisition data is expressed as the number of acquisition numbers. In Free Run mode, it is expressed as the number of sample points.

Parameters
- hCh: A channel handle
- size: Where to store the information about data storage capacity

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.11 Auto Recording Condition Settings and Queries

Query or Set Auto Recording Destination

```c
int SxSetRecDest(SX_HNDL_GROUP hGrp, int dest);
```

**Description**
Sets the auto recording destination of the unit group specified by hGrp to the destination specified by dest.

**Parameters**
- **hGrp**: A unit group handle
- **dest**: SX_REC_DEST_PC: The PC hard disk
  - SX_REC_DEST_UNIT: The unit’s hard disk
  - SX_REC_DEST_PC_UNIT: The PC and the unit’s hard disks.

**Cannot be set during synchronous operation.**

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecDest(SX_HNDL_GROUP hGrp, int *dest);
```

**Description**
Queries the auto recording destination of the unit group specified by hGrp.

**Parameters**
- **hGrp**: A unit group handle
- **dest**: Where to store the auto recording destination information
  - SX_REC_DEST_PC: The PC hard disk
  - SX_REC_DEST_UNIT: The unit’s hard disk
  - SX_REC_DEST_PC_UNIT: The PC and the unit’s hard disks.

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Start Condition

```c
int SxSetRecStartCond(SX_HNDL_GROUP hGrp, int startCond);
```

**Description**
Sets the recording start condition of the unit group specified by hGrp to the condition specified by startCond.

**Parameters**
- **hGrp**: A unit group handle
- **startCond**: SX_REC_START_IMMEDIATE: Start immediately
  - SX_REC_START_TIME: Start at a set time
  - SX_REC_START_ALARM: Start when an alarm occurs
  - SX_REC_START_TRIG_RISE: Start on the rising edge of an external trigger signal
  - SX_REC_START_TRIG_FALL: Start on the falling edge of an external trigger signal

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.11 Auto Recording Condition Settings and Queries

```c
int SxGetRecStartCond(SX_HNDL_GROUP hGrp, int *startCond);
ERR GetRecStartCond( HNDL hGrp, ref REC_START recCond );
```

**Description**
Queries the recording start condition of the unit group specified by hGrp.

**Parameters**
- `hGrp` A unit group handle
- `startCond` Where to store the recording start condition
  - `SX_REC_START_IMMIDIATE`: Start immediately
  - `SX_REC_START_TIME`: Start at a set time
  - `SX_REC_START_ALARM`: Start when an alarm occurs
  - `SX_REC_START_TRIG_RISE`: Start on the rising edge of an external trigger signal
  - `SX_REC_START_TRIG_FALL`: Start on the falling edge of an external trigger signal

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**Query or Set Recording Start Time**
```c
int SxSetRecStartTime(SX_HNDL_GROUP hGrp, char *datetime);
ERR SetRecStartTime( HNDL hGrp, string datetime );
```

**Description**
Sets the recording start time of the unit group specified by hGrp to the date and time specified by datetime. This setting is only valid when the recording start condition is set to `SX_REC_START_TIME`.

**Parameters**
- `hGrp` A unit group handle
- `datetime` Where to store the date and time text string
  - Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecStartTime(SX_HNDL_GROUP hGrp, char *datetime);
ERR GetRecStartTime( HNDL hGrp, ref string datetime );
```

**Description**
Queries the recording start time of the unit group specified by hGrp.

**Parameters**
- `hGrp` A unit group handle
- `datetime` Where to store the date and time text string (20 bytes or more)
  - Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.11 Auto Recording Condition Settings and Queries

Query or Set Recording Stop Condition

int SxSetRecStopCond(SX_HNDL_GROUP hGrp, int stopCond);
ERR SetRecStopCond( HNDL hGrp, REC_STOP recCond ) ;

Description
Sets the recording stop condition of the unit group specified by hGrp to the condition specified by stopCond.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrp</td>
<td>A unit group handle</td>
</tr>
<tr>
<td>stopCond</td>
<td>SX_REC_STOP_CONTINUOUS: Continue until a recording stop command is received.</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TIME: Stop at the set time. (This setting is invalid when the recording start condition is set to SX_REC_START_ALARM).</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_SPAN: Stop after the specified recording time (when the clock source is the internal clock). Stop after the specified number of points have been recorded (when the clock source is an external clock).</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_ALARM: Stop when an alarm occurs (when the recording start condition is set to SX_REC_START_ALARM). Stop when an alarm is released (when the recording start condition is not SX_REC_START_ALARM).</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TRIG_RISE: Stop on the rising edge of an external trigger signal.</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TRIG_FALL: Stop on the falling edge of an external trigger signal.</td>
</tr>
</tbody>
</table>

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecStopCond(SX_HNDL_GROUP hGrp, int *stopCond);
ERR GetRecStopCond( HNDL hGrp, ref REC_STOP recCond ) ;

Description
Queries the recording stop condition of the unit group specified by hGrp.

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hGrp</td>
<td>A unit group handle</td>
</tr>
<tr>
<td>stopCond</td>
<td>Where to store the recording stop condition SX_REC_STOP_CONTINUOUS: Continue until a recording stop command is received.</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TIME: Stop at the set time. (This setting is invalid when the recording start condition is set to SX_REC_START_ALARM.)</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_SPAN: Stop after the specified recording time (when the clock source is the internal clock). Stop after the specified number of points have been recorded (when the clock source is an external clock).</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_ALARM: Stop when an alarm occurs (when the recording start condition is set to SX_REC_START_ALARM). Stop when an alarm is released (when the recording start condition is not SX_REC_START_ALARM).</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TRIG_RISE: Stop on the rising edge of an external trigger signal.</td>
</tr>
<tr>
<td></td>
<td>SX_REC_STOP_TRIG_FALL: Stop on the falling edge of an external trigger signal</td>
</tr>
</tbody>
</table>

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.
3.11 Auto Recording Condition Settings and Queries

Query or Set Recording Stop Time
int SxSetRecStopTime(SX_HNDL_GROUP hGrp, char *datetime);

Description
Sets the recording stop time of the unit group specified by hGrp to the date and time
specified by datetime. This setting is only valid when the recording stop condition is set
to SX_REC_STOP_TIME.

Parameters
hGrp    A unit group handle
datetime Where to store the date and time text string

Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.

int SxGetRecStopTime(SX_HNDL_GROUP hGrp, char *datetime);

Description
Queries the recording stop time of the unit group specified by hGrp.

Parameters
hGrp    A unit group handle
datetime Where to store the date and time text string (20 bytes or more)

Format: "YYYY/MM/DD-hh:mm:ss" (19 characters + 1 null character)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.

Query or Set Recording Time
int SxSetRecSpan(SX_HNDL_GROUP hGrp, double second);

Description
Sets the recording time of the unit group specified by hGrp to the length in seconds
specified by the second parameter.
This setting is only valid when the clock source is the internal clock and the recording
stop condition is set to SX_REC_STOP_SPAN.

Parameters
hGrp    A unit group handle
second The length of the recording time in seconds

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.

int SxGetRecSpan(SX_HNDL_GROUP hGrp, double *second);

Description
Queries the recording time (in seconds) of the unit group specified by hGrp.
This setting is only valid when the clock source is the internal clock and the recording
stop condition is set to SX_REC_STOP_SPAN.

Parameters
hGrp    A unit group handle
second Where to store the length of the recording time (in seconds)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.
Query or Set Number of Points to Record

```c
int SxSetRecExtClkPoints(SX_HNDL_GROUP hGrp, int points);
ERR SetRecExtClkPoints( HNDL hGrp, int points );
```

**Description**

Sets the number of points that the unit group specified by hGrp will record.
This setting is only valid when the clock source is the external clock and the recording stop condition is set to SX_REC_STOP_SPAN.

**Parameters**

- **hGrp** A unit group handle
- **points** The number of points to record (the number can be set to a value from 10 to 10,000,000, but it must not exceed the recording interval).

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecExtClkPoints(SX_HNDL_GROUP hGrp, int *points);
ERR GetRecExtClkPoints( HNDL hGrp, ref int points );
```

**Description**

Queries the number of points to record in the unit group specified by hGrp.
This setting is only valid when the clock source is the external clock and the recording stop condition is set to SX_REC_STOP_SPAN.

**Parameters**

- **hGrp** A unit group handle
- **points** Where to store the number of points to record

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Recording Interval Mode

```c
int SxSetRecIntervalMode(SX_HNDL_GROUP hGrp, int mode);
ERR SetRecIntervalMode( HNDL hGrp, BOOL mode );
```

**Description**

Sets the recording interval mode of the unit group specified by hGrp to the mode specified by the mode parameter. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

**Parameters**

- **hGrp** A unit group handle
- **mode**
  - SX_REC_INTERVAL_OFF: Do not record at a set interval
  - SX_REC_INTERVAL_ON: Record at a set interval

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecIntervalMode(SX_HNDL_GROUP hGrp, int *mode);
ERR GetRecIntervalMode( HNDL hGrp, ref BOOL mode );
```

**Description**

Queries the recording interval mode of the unit group specified by hGrp.
This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

**Parameters**

- **hGrp** A unit group handle
- **mode**
  - SX_REC_INTERVAL_OFF: Do not record at a set interval
  - SX_REC_INTERVAL_ON: Record at a set interval

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Recording Interval
int SxSetRecInterval(SX_HNDL_GROUP hGrp, double second);
ERR SetRecInterval( HNDL hGrp, double second );

Description
Sets the recording interval of the unit group specified by hGrp to the length in seconds
specified by the second parameter.
This setting is only valid when the clock source is the internal clock, the recording stop
condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_
REC_INTERVAL_ON.

Parameters
- hGrp: A unit group handle
- second: The recording interval, in seconds. The interval can be set to
  the second to a value from 1 to 86,400 seconds.

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecInterval(SX_HNDL_GROUP hGrp, double *second);
ERR GetRecInterval( HNDL hGrp, ref double second );

Description
Queries the recording interval (in seconds) of the unit group specified by hGrp.
This setting is only valid when the clock source is the internal clock, the recording stop
condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_
REC_INTERVAL_ON.

Parameters
- hGrp: A unit group handle
- second: Where to store the recording interval (in seconds)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Recording Interval Points
int SxSetRecExtClkInterval(SX_HNDL_GROUP hGrp, int interval);
ERR SetRecExtClkInterval( HNDL hGrp, int interval );

Description
Sets the number of recording interval points of the unit group specified by hGrp to the
number of sample points specified by the interval parameter.
This setting is only valid when the clock source is the external clock, the recording stop
condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_
REC_INTERVAL_ON.

Parameters
- hGrp: A unit group handle
- interval: The number of recording interval points (can be set to a value from 10 to
  1,000,000)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecExtClkInterval(SX_HNDL_GROUP hGrp, int *interval);
ERR GetRecExtClkInterval( HNDL hGrp, ref int interval );

Description
Queries the number of recording interval points in the unit group specified by hGrp.
This setting is only valid when the clock source is the external clock, the recording stop
condition is set to SX_REC_STOP_SPAN, and the recording interval mode is set to SX_
REC_INTERVAL_ON.

Parameters
- hGrp: A unit group handle
- interval: Where to store the number of recording interval points

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.
3.11 Auto Recording Condition Settings and Queries

Query or Set Record Count
int SxSetRecTimes(SX_HNDL_GROUP hGrp, int recNum);
ERR SetRecTimes( HNDL hGrp, int recNum );

Description
Sets the record count of the unit group specified by hGrp to value specified by recNum. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters
- hGrp: A unit group handle
- recNum: The record count (Can be set to a value from 0 to 100,000. A value of 0 indicates no limit.)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecTimes(SX_HNDL_GROUP hGrp, int *recNum);
ERR GetRecTimes( HNDL hGrp, ref int recNum );

Description
Queries the record count of the unit group specified by hGrp. This setting is only valid when the recording stop condition is set to SX_REC_STOP_SPAN.

Parameters
- hGrp: A unit group handle
- recNum: Where to store the record count (0 indicates no limit)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

Query or Set Recording Destination Folder (on the PC)
int SxSetRecFileFolder(SX_HNDL_GROUP hGrp, char *path);
ERR GetRecFileFolder( HNDL hGrp, ref string path );

Description
Sets the PC auto recording destination folder of the unit group specified by hGrp to the path specified by the path parameter. The maximum number of characters for the path parameter is 510. This setting is valid when the auto recording destination is set to SX_REC_DEST_PC or SX_REC_DEST_PCUNIT.

Parameters
- hGrp: A unit group handle
- path: The auto recording destination folder on the PC (510 characters or less) Example: "C:\SL1000"

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

int SxGetRecFileFolder(SX_HNDL_GROUP hGrp, char *path);
ERR GetRecFileFolder( HNDL hGrp, ref string path );

Description
Queries the PC auto recording destination folder of the unit group specified by hGrp. This setting is valid when the auto recording destination is set to SX_REC_DEST_PC or SX_REC_DEST_PCUNIT.

Parameters
- hGrp: A unit group handle
- path: Where to store the PC auto recording destination folder (requires 512 or more bytes of memory)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Auto Naming

```c
int SxSetRecFileAutoNaming(SX_HNDL_GROUP hGrp, int type);
```  
**Description**
Sets the auto naming of the unit group specified by hGrp to the condition specified by the type parameter.

**Parameters**
- **hGrp**: A unit group handle
- **type**: SX_REC_AUTONAME_DATE: Date and time
  - Cannot be set during synchronous operation.
- **type**: SX_REC_AUTONAME_NUM: Sequential numbering

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecFileAutoNaming(SX_HNDL_GROUP hGrp, int *type);
```  
**Description**
Queries the auto naming setting of the unit group specified by hGrp.

**Parameters**
- **hGrp**: A unit group handle
- **type**: Where to store the auto naming setting
  - SX_REC_AUTONAME_DATE: Date and time
  - SX_REC_AUTONAME_NUM: Sequential numbering

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set File Name

```c
int SxSetRecFileName(SX_HNDL_GROUP hGrp, char *fileName);
```  
**Description**
Sets the auto recording file name of the unit group specified by hGrp to the string specified by fileName. The maximum number of characters for the fileName parameter (for the PC) is 255, not including the null character.

This setting is valid when auto naming is set to SX_REC_AUTONAME_NUM.

**Parameters**
- **hGrp**: A unit group handle
- **fileName**: The file name (255 characters or less)
  - Example: "TEST"

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecFileName(SX_HNDL_GROUP hGrp, char *fileName);
```  
**Description**
Queries the auto recording file name of the unit group specified by hGrp.

**Parameters**
- **hGrp**: A unit group handle
- **fileName**: Where to store the file name (requires 256 or more bytes of memory)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.11 Auto Recording Condition Settings and Queries

**Query or Set File Order**

```c
int SxSetRecFileOrder(SX_HNDL_GROUP hGrp, int fileOrder);
ERR SetRecFileOrder( HNDL hGrp, REC_FILEORDER fileOrder );
```

*Description*
Sets the file order of the unit group specified by hGrp to the condition specified by fileOrder.

*Parameters*
- `hGrp`: A unit group handle
- `fileOrder`: SX_REC_FILEORDER_SEQUENTIAL: Sequential  
  SX_REC_FILEORDER_CYCLIC: Cyclic

*Return Value*
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecFileOrder(SX_HNDL_GROUP hGrp, int *fileOrder);
ERR GetRecFileOrder( HNDL hGrp, ref REC_FILEORDER fileOrder );
```

*Description*
Queries the file order of the unit group specified by hGrp.

*Parameters*
- `hGrp`: A unit group handle
- `fileOrder`: Where to store the file order
  SX_REC_FILEORDER_SEQUENTIAL: Sequential  
  SX_REC_FILEORDER_CYCLIC: Cyclic

*Return Value*
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**Query or Set File Count Limit**

```c
int SxSetRecCyclicFiles(SX_HNDL_GROUP hGrp, int fileNum);
ERR SetRecCyclicFiles( HNDL hGrp, int fileNum );
```

*Description*
Sets the file count limit of the unit group specified by hGrp to the value specified by fileNum. This setting is valid when the file order is set to SX_REC_FILEORDER_CYCLIC.

*Parameters*
- `hGrp`: A unit group handle
- `fileNum`: The file count limit (1 to 1000)

*Return Value*
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

```c
int SxGetRecCyclicFiles(SX_HNDL_GROUP hGrp, int *fileNum);
ERR GetRecCyclicFiles( HNDL hGrp, ref int fileNum );
```

*Description*
Queries the file count limit of the unit group specified by hGrp. This setting is valid when the file order is set to SX_REC_FILEORDER_CYCLIC.

*Parameters*
- `hGrp`: A unit group handle
- `fileNum`: Where to store the file count limit

*Return Value*
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Comment

int SxSetRecFileComment(SX_HNDL_GROUP hGrp, char *comment);

ERR SetRecFileComment( HNDL hGrp, string comment );

Description
Sets the comment to be saved in the auto recording file of the unit group specified
by hGrp to the string specified by the comment parameter. The maximum number of
characters for the comment parameter is 250, not including the null character.

Parameters
hGrp A unit group handle
comment The comment (250 characters or less)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.

int SxGetRecFileComment(SX_HNDL_GROUP hGrp, char *comment);

ERR GetRecFileComment( HNDL hGrp, ref string comment );

Description
Queries the comment to be saved in the auto recording file of the unit group specified by
hGrp.

Parameters
hGrp A unit group handle
comment Where to store the comment (requires 251 or more bytes of memory)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.

Get Information About the File Being Recorded (on the PC)

int SxGetRecCurrentFileInfo(SX_HNDL_UNIT hUnit, char *fileName, UINT *fileSize);

ERR GetRecCurrentFileInfo( HNDL hUnit, ref string filename, ref uint filesize );

Description
Retrieves the information of the file that contains the data from the unit specified by hUnit
that is being automatically recorded (saved) on the PC.

Parameters
hUnit A unit handle
fileName Where to store the file path (requires 512 or more bytes of memory)
fileSize Where to store the file size (in bytes)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not
succeed.

Get Information about Files That Have Been Recorded (on the PC)

int SxGetRecLastFileInfo(SX_HNDL_UNIT hUnit, char *fileName, UINT *fileSize);

ERR GetRecLastFileInfo( HNDL hUnit, ref string filename, ref uint filesize );

Description
Retrieves the file information of the file that contains the data from the unit specified by hUnit
that was automatically recorded and closed on the PC last.

Parameters
hUnit A unit handle
fileName Where to store the file path (requires 512 or more bytes of memory)
fileSize Where to store the file size (in bytes)

Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.
3.12 Measurement Controls

Start Measurement
int SxAcqStart( SX_HNDL_GROUP hGrp );
ERR AcqStart( HNDL hGrp);

Description
Causes the units in the unit group specified by hGrp to start measuring.

Parameters
hGrp A unit group handle

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Stop Measurement
int SxAcqStop( SX_HNDL_GROUP hGrp );
ERR AcqStop( HNDL hGrp);

Description
Causes the units in the unit group specified by hGrp to stop measuring.
If you execute this function during auto recording, the auto recording will also stop.

Parameters
hGrp A unit group handle

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Execute Latch (in Free Run Mode)
int SxAcqLatch( SX_HNDL_GROUP hGrp, int *point );
int SxAcqLatchL( SX_HNDL_GROUP hGrp, __int64 *point );
int SxAcqLatchD( SX_HNDL_GROUP hGrp, double *point );
ERR AcqLatch( HNDL hGrp, ref long point );
ERR AcqLatch( HNDL hGrp );

Description
Stores to num the number of sample points that have been sampled since measurement began by the units that belong to the unit group specified by hGrp. The number of sample points retrieved is the number of sample points sampled by measuring group 1.
This function is only valid in Free Run mode.
To retrieve the waveform data between the previous latch and the current latch, set the AcqNo parameter of SxGetAcqData() or SxGetAcqDataEx() to 0.
If the number of sample points exceeds 2,147,483,647, use SxAcqLatchL() or SxAcqLatchD(). SxAcqLatchL() cannot be used in VB6.

Parameters
hGrp A unit group handle
point Where to store the number of sample points that have been sampled by measuring group 1 since the start of measurement at the time the function is executed.

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Confirm Measure/Save Operation
int SxIsRun( SX_HNDL_GROUP hGrp, int* status );
ERR IsRun( HNDL hGrp, ref int status );

Description
Determines whether the units that belong to the group specified by hGrp are currently measuring, saving, or waiting for a trigger.

Parameters
hGrp A unit group handle
status Where to store the measurement condition
b0: Measuring  b1: Saving  b2: Waiting for a trigger

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Execute Manual Trigger
int SxExecManualTrig( SX_HNDL_GROUP | SX_HNDL_UNIT hAny );
ERR ExecManualTrig( HNDL hAny );

Description
Executes a manual trigger that affects the units that belong to the object specified by hAny.

Parameters
hAny Any kind of handle

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.13 Auto Recording Controls

Start Recording

\[
\text{int SxRecStart( SX_HNDL_GROUP hGrp );}
\]

\[
\text{ERR RecStart( HNDL hGrp );}
\]

- **Description**
  
  Causes the units in the unit group specified by hGrp to start auto recording. If you execute this function when measurement has not started, measurement will start. Use this function after setting the auto recording condition with the set auto recording condition function.

- **Parameters**
  
  hGrp: A unit group handle

- **Return Value**
  
  Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Stop Recording

\[
\text{int SxRecStop( SX_HNDL_GROUP hGrp );}
\]

\[
\text{ERR RecStop( HNDL hGrp );}
\]

- **Description**
  
  Causes the units in the unit group specified by hGrp to stop auto recording. Measurement will continue even after this function is executed.

- **Parameters**
  
  hGrp: A unit group handle

- **Return Value**
  
  Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Divide File

\[
\text{int SxRecDivide( SX_HNDL_GROUP hGrp );}
\]

\[
\text{ERR RecDivide( HNDL hGrp );}
\]

- **Description**
  
  If the units that belong to the unit group specified by hGrp are auto recording in Free Run mode, this function closes the currently recorded file and continues recording on a new file. Cannot be executed during synchronous operation.

- **Parameters**
  
  hGrp: A unit group handle

- **Return Value**
  
  Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.14 Acquisition and Deletion of Measured Data

Get Acquisition Data Information (Constants for Converting Physical Values)

```c
int SxGetChannelInfo( SX_HNDL_CH hCh, SX_INFO_CH *inf );
```

**Description**
Retrieves the acquisition data information for converting the measured data from the channel specified by hCh to physical values.

**Parameters**
- **hCh**
  A channel handle
- **inf**
  Where to store the retrieved data information

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Latch Interval Sample Points

```c
int SxGetLatchLength( SX_HNDL_MEASGRP | SX_HNDL_GROUP hAny, int *len );
```

**Description**
Returns the latch interval sample points. If hAny is a measuring group, the function returns the number of sample points in the measuring group. If hAny is a unit group, the function returns the number of sample points in measuring group 1 in the unit group. This function is valid in Free Run mode.

**Parameters**
- **hAny**
  Any kind of handle
- **len**
  Where to store the number latch interval sample points

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Latest Acquisition Number

```c
int SxGetLatestAcqNo( SX_HNDL_GROUP hGrp, int *acqNo );
i accordance with the acquisition number. If the number of acquisitions exceeds 2,147,483,647, use SxGetLatestAcqNoL() or SxGetLatestAcqNoD(). SxGetLatestAcqNoL() cannot be used in VB6.

**Parameters**
- **hGrp**
  A unit group handle
- **acqNo**
  Where to store the latest acquisition number

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.14 Acquisition and Deletion of Measured Data

Get Waveform Data

```c
int SxGetAcqData( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqNo, void* buf, int blen, int *rlen);
int SxGetAcqDataL( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, __int64 acqNo, void* buf, int blen, int *rlen);
int SxGetAcqDataD( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, double acqNo, void* buf, int blen, int *rlen);
```

**ERR GetAcqData( HNDL hAny, long acqNo, ref any[] buf, int blen, ref int rlen );**

**ERR GetAcqData( HNDL hAny, long acqNo, ref any[] buf, ref int rlen );**

**Description**
Stores the waveform data with the acquisition number specified by acqNo to the memory location specified by buf.

The values stored in buf are A/D converted values. To convert the data into physical values, the vResolution and vOffset acquisition data information values are necessary.

If the waveform data from multiple channels is stored, the data will be stored in block format starting with the lowest numbered channel's data.

Use blen to specify the size in bytes of buf.

The actual stored byte size will be stored in rlen.

If hAny is a channel handle, only the waveform data for that channel will be stored.

If hAny is a measuring group, the waveform data of all of the channels in the measuring group will be stored.

If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqDataL() or SxGetAcqDataD(). SxGetAcqDataL() cannot be used in VB6.

**Parameters**

- **hAny**: Any kind of handle
- **acqNo**: An acquisition number
  - Positive number: An absolute acquisition number
  - Negative number: A relative number with –1 being the most recent acquisition.
  - Zero: All of the history (in Triggered mode)
  - The latch interval (in Free Run mode)
- **buf**: Where to store the waveform data
- **blen**: The size of the memory area where the waveform data is stored (in bytes)
- **rlen**: Where to store the size of the stored waveform data

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Get Waveform Data Segment
int SxGetAcqDataEx( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, int acqNo, int start, int count,
void* buf, int blen, int *rlen );
int SxGetAcqDataExL( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, __int64 acqNo, __int64 start,
int count, void* buf, int blen, int *rlen );
int SxGetAcqDataExD( SX_HNDL_CH | SX_HNDL_MEASGRP hAny, double acqNo, double start,
int count, void* buf, int blen, int *rlen );
ERR GetAcqDataEx( HNDL hAny, long acqNo, long start, int count, ref any[] buf, int blen, ref int
rlen );
ERR GetAcqDataEx( HNDL hAny, long acqNo, long start, int count, ref any[] buf, ref int rlen );

Description
Stores the waveform data with the acquisition number specified by acqNo to the memory
location specified by buf. The values stored in buf are A/D converted values. To convert
the data into physical values, the vResolution and vOffset acquisition data information
values are necessary.
If the waveform data from multiple channels is stored, the data will be stored in block
format starting with the lowest numbered channel’s data.
Specify the beginning of the waveform data segment that you want to retrieve using the
start parameter. Specify the value using the sampling units of measuring group 1.
Specify the amount waveform data that you want to retrieve using the count parameter.
Specify the value using the sampling units of measuring group 1. If you specify 0 for
count, all of the data from the beginning of the segment to the end of the waveform data
will be retrieved.
Use blen to specify the size in bytes of buf. The actual stored byte size will be stored in
rlen.
If hAny is a channel handle, only the waveform data for that channel will be stored.
If hAny is a measuring group, the waveform data of all of the channels in the measuring
group will be stored.
If the number of acquisitions and measurement points exceeds 2,147,483,647, use
SxGetAcqDataExL() or SxGetAcqDataExD(). SxGetAcqDataExL() cannot be used in
VB6.

Parameters
hAny Any kind of handle
acqNo An acquisition number
Positive number: An absolute acquisition number
Negative number: A relative number with –1 being the most recent
acquisition.
Zero: All of the history (in Triggered mode)
The latch interval (in Free Run mode)
start The start of the data segment to be retrieved (in the sampling units of
measuring group 1, starting from 0)
count The length of the data segment to be retrieved (in the sampling units of
measuring group 1, starting from 0)
buf Where to store the waveform data
blen The size of the memory area where the waveform data is stored (in bytes)
rlen Where to store the size of the stored waveform data

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.
Get Data Acquisition Time

int SxGetAcqTime( SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, int acqNo, char *datetime );

int SxGetAcqTimeL( SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, __int64 acqNo, char *datetime);

int SxGetAcqTimeD( SX_HNDL_MEASGRP | SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH hAny, double acqNo, char *datetime);

ERR GetAcqTime( HNDL hAny, long acqNo, ref string datetime );

Description
Retrieves the trigger time of the waveform data with the acquisition number specified by
acqNo in the unit that belongs to the object specified by hAny (if hAny is a unit group or
measuring group, the master unit is used). The acqNo specification is invalid in Free Run
mode, and the time when measurement started is acquired.
Allocate at least 27 bytes of memory area for datetime.
If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqTimeL() or
SxGetAcqTimeD(). SxGetAcqTimeL() cannot be used in VB6.

Parameters
hAny Any kind of handle
acqNo An acquisition number (this parameter is invalid in Free Run mode)
Positive number: An absolute acquisition number
Negative number: A relative number with –1 being the most recent
acquisition.
datetime Where to store the date and time text string
Format: "YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Get Phase Difference between Measuring Groups

```c
int SxGetAcqDelay( SX_HNDL_MEASGRP hMgrp, int acqNo, int *delay );
int SxGetAcqDelayL( SX_HNDL_MEASGRP hMgrp, __int64 acqNo, int *delay );
int SxGetAcqDelayD( SX_HNDL_MEASGRP hMgrp, double acqNo, int *delay );
ERR GetAcqDelay( HNDL hAny, long acqNo, ref int delay );
```

**Description**
Retrieves the difference in measurement points between (1) the first sample point in the waveform data with the acquisition number specified by acqNo in the measurement group specified by hMgrp and (2) the first sample point in measuring group 1 sample interval. The function returns zero if there is no difference. If the number of acquisitions exceeds 2,147,483,647, use SxGetAcqDelayL() or SxGetAcqDelayD(). SxGetAcqDelayL() cannot be used in VB6.

**Parameters**
- **hMgrp**: A measuring group handle
- **acqNo**: An acquisition number (this parameter is invalid in Free Run mode)
  - Positive number: An absolute acquisition number
  - Negative number: A relative number with -1 being the most recent acquisition.
- **delay**: Where to store the difference between the sampling start points of measuring group 1 and the specified waveform data (from 0).

**Return Value**
Returns zero if the function succeeds and a nonzero error code if it does not.

Create WDF File

```c
int SxSaveAcqData( SX_HNDL_UNIT hUnit, int acqNo, char *fileName );
int SxSaveAcqDataL( SX_HNDL_UNIT hUnit, __int64 acqNo, char *fileName);
int SxSaveAcqDataD( SX_HNDL_UNIT hUnit, double acqNo, char *fileName);
ERR SaveAcqData( HNDL hUnit, long acqNo, string fileName );
```

**Description**
Creates a WDF file with the name specified by fileName and saves waveform data of the acquisition number specified by acqNo of the unit specified by hUnit. The channels that are saved those whose recording is set to On using the SxSetRecSwitch() function. A positive acqNo number indicates an absolute acquisition number. A negative acqNo number indicates history data with -1 corresponding to the latest data, -2 corresponding to the next latest data, and so on. When acqNo is set to zero, all history data are saved. You cannot save all history data during measurement. The extension is .wdf. The extension is added automatically, so it does not need to be included in fileName. This function is valid in Triggered mode. If the number of acquisitions exceeds 2,147,483,647, use SxSaveAcqDataL() or SxSaveAcqDataD(). SxSaveAcqDataL() cannot be used in VB6.

**Parameters**
- **hUnit**: A unit handle
- **acqNo**: An acquisition number
  - Positive number: The specified number
  - Negative number: A relative number with -1 being the most recent acquisition.
  - Zero: All of the history (in Triggered mode)
- **fileName**: The file name

**Return Value**
Returns zero if the function succeeds and a nonzero error code if it does not.
### Create WDF File (from Waveform Data Segment)

```c
int SxSaveAcqDataEx(SX_HNDL_UNIT hUnit, int acqNo, int start, int count,
char *fileName);

int SxSaveAcqDataExL(SX_HNDL_UNIT hUnit, __int64 acqNo, int start, int count,
char *fileName);

int SxSaveAcqDataExD(SX_HNDL_UNIT hUnit, double acqNo, int start, int count,
char *fileName);

ERR SaveAcqDataEx(HNDL hUnit, long acqNo, long start, int count, string filename);
```

**Description**

Creates a WDF file with the name specified by `fileName` and saves waveform data of the acquisition number specified by `acqNo` of the unit specified by `hUnit`. The channels that are saved are those whose recording is set to On using the SxSetRecSwitch() function. Specify the beginning of the waveform data segment that you want to save using the `start` parameter. Specify the value using the sampling units of measuring group 1. Specify the amount of waveform data that you want to save using the `count` parameter. Specify the value using the sampling units of measuring group 1. If you specify 0 for `count`, all of the data from the beginning of the segment to the end of the waveform data will be saved.

A positive `acqNo` number indicates an absolute acquisition number. A negative `acqNo` number indicates history data with `-1` corresponding to the latest data, `-2` corresponding to the next latest data, and so on. When `acqNo` is set to zero, all history data are saved. You cannot save all history data during measurement.

The extension is `.wdf`. The extension is added automatically, so it does not need to be included in `fileName`. This function is valid in Triggered mode.

If the number of acquisitions exceeds 2,147,483,647, use SxSaveAcqDataExL() or SxSaveAcqDataExD(). SxSaveAcqDataExL() cannot be used in VB6.

**Parameters**

- **hUnit**: A unit handle
- **acqNo**: An acquisition number
  - Positive number: The specified number
  - Negative number: A relative number with `-1` being the most recent acquisition.
  - Zero: All of the history (in Triggered mode)
- **fileName**: The file name

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Get Instantaneous Values

int SxGetCurrentData( SX_HNDL_GROUP | SX_HNDL_UNIT | SX_HNDL_MOD | SX_HNDL_CH | SX_HNDL_MEASGRP | SX_HNDL_CHGRP hAny, void* buf, int blen, int *rlen );

ERR GetCurrentData( HNDL hAny, ref any[] buf, int blen, ref int rlen );
ERR GetCurrentData( HNDL hAny, ref any[] buf, ref int rlen );
ERR GetCurrentData( HNDL hAny, ref any buf );

Description
Stores instantaneous values (the most recently measured values) in the memory area specified by buf.

The values stored in buf are A/D converted values. To convert the data into physical values, the vResolution and vOffset acquisition data information values are necessary.

If the instantaneous values from multiple channels is stored, the data will be stored in block format starting with the lowest numbered channel’s data.

Use blen to specify the size in bytes of buf.

The actual stored byte size will be stored in rlen.

If hAny is a channel handle, only the instantaneous values for that channel will be stored.
If hAny is a module handle, all of the instantaneous values of the channels in that module will be stored.
If hAny is a unit handle, all of the instantaneous values of the channels in that unit will be stored.
If hAny is a unit group handle, all of the instantaneous values of the channels in that unit group will be stored.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAny</td>
<td>Any kind of handle</td>
</tr>
<tr>
<td>buf</td>
<td>Where to store the instantaneous values</td>
</tr>
<tr>
<td>blen</td>
<td>The size of the memory area where the instantaneous values are stored (in bytes)</td>
</tr>
<tr>
<td>rlen</td>
<td>Where to store the size of the stored instantaneous values</td>
</tr>
</tbody>
</table>

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Waveform Data

int SxClearAcqData( SX_HNDL_GROUP | SX_HNDL_UNIT hAny );

ERR SaveSetup( HNDL hUnit, string fileName );

Description
Deletes all of the waveform data (clears the entire history) of the units that belong to the object specified by hAny.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hAny</td>
<td>Any kind of handle</td>
</tr>
</tbody>
</table>

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.15 Setup Data Access

Save Setup Data
int SxSaveSetup(SX_HNDL_UNIT hUnit, char *fileName);

ERR SaveSetup( HNDL hUnit, string fileName );

Description
Saves the setup data of the unit specified by hUnit to the file (on the PC) specified by fileName.

Parameters
hUnit   A unit handle
fileName A file name

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Load Setup Data
int SxLoadSetup(SX_HNDL_UNIT hUnit, char *fileName);

ERR LoadSetup( HNDL hUnit, string fileName );

Description
Loads the setup data stored to the file (on the PC) specified by fileName to the unit specified by hUnit.

Parameters
hUnit   A unit handle
fileName A file name

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Initialize Setup Data
int SxInitSetup( SX_HNDL_GROUP | SX_HNDL_UNIT hAny );

ERR InitSetup( HNDL hAny );

Description
Returns the setup data on the units specified by hAny to the factory default settings of those units.
If hAny is a unit group, the function initializes the setup data of all of the units in that group.

Parameters
hAny    Any kind of handle

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.16 System-Related Functions

Set Unit Group Name
int SxSetGroupName( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *name);

ERR SetGroupName( HNDL hAny, ref string name );

Description
Assigns the unit group name specified by name to the units that belong to the object
specified by hAny. An error occurs if you specify a character string that is greater than or
equal to 32 characters in length.

Parameters
hAny Any kind of handle
name Unit group name (up to 32 characters, including the null character)

Return Value
int Returns zero if the function succeeds and a nonzero error code if the
function does not succeed.

Set Unit Name
int SxSetUnitName( SX_HNDL_UNIT hUnit, char *name);

ERR SetUnitName( HNDL hUnit, ref string name );

Description
Assign the unit name specified by name to the unit specified by hUnit. An error occurs if
you specify a character string that is greater than or equal to 32 characters in length.

Parameters
hUnit A unit handle
name Unit name (up to 32 characters, including the null character)

Return Value
int Returns zero if the function succeeds and a nonzero error code if the
function does not succeed.

Execute Calibration
int SxExecCal( SX_HNDL_UNIT hUnit, int *result );

ERR ExecCal( HNDL hUnit, ref int result );

Description
Calibrates the unit specified by hUnit. This function returns after calibration is complete.

Parameters
hUnit A unit handle
result Where to store the results of calibration
    Zero: Succeeded
    Nonzero: Failed

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not
succeed.
3.16 System-Related Functions

Execute Self Test
int SxExecSelftest( SX_HNDL_UNIT hUnit, int *result );
ERR ExecSelftest( HNDL hUnit, ref int result );

Description
Executes a self test on the unit specified by hUnit.
Specify the item you want to test using *result. The test result is returned in *result.

Parameters
- hUnit: A unit handle
- result: Where to store the item to test and the result of the self test
  The item to test and the results of the test are indicated using the logical sum of these constants
  - SX_SELFTEST_ACQMEM: Waveform memory error
  - SX_SELFTEST_SYSMEM: System error
  - SX_SELFTEST_BACKUPMEM: Backup memory error
  - SX_SELFTEST_HDD: Internal hard disk error
  If you specify 0, all items are tested.
  If there are no errors, 0 will be stored.

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get Unit CPU Temperature
int SxGetCpuTemperature( SX_HNDL_UNIT hUnit, int *temperature );
ERR GetCpuTemperature( HNDL hUnit, ref int temperature );

Description
Retrieves the CPU temperature of the unit specified by hUnit.

Parameters
- hUnit: A unit handle
- temperature: Where to store the temperature (in °C)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Query or Set Unit Clock

#### Description
Sets the system clocks of the units that belong to the object specified by hAny. During synchronous operation, the time cannot be set on the SL1000 unit alone.

#### Parameters
- **hAny**: Any kind of handle
- **datetime**: A date and time text string (the time can be specified to the microsecond)
  - Format: "YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character)

#### Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

#### Example

```c
int SxSetSystemClock( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *datetime );
ERR SetSystemClock( HNDL hAny, string datetime );
```

### Get System Clock

#### Description
Queries the system clock of the unit that belongs to the object specified by hAny.
Allocate at least 27 bytes of memory area for datetime.
If hAny is a unit group, the function queries the master unit’s system clock.

#### Parameters
- **hAny**: Any kind of handle
- **datetime**: Where to store the date and time text string (the time can be specified to the microsecond)
  - Format: "YYYY/MM/DD-hh:mm:ss.uuuuuu" (26 characters + 1 null character)

#### Return Value
Returns zero if the function succeeds and a nonzero error code if it does not.

#### Example

```c
int SxGetSystemClock( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *datetime );
ERR GetSystemClock( HNDL hAny, ref string datetime );
```
3.16 System-Related Functions

Query or Set DHCP

int SxSetEthernetDHCP( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode );
ERR SetEthernetDHCP( HNDL hAny, int mode );

Description
Sets the DHCP client of the units that belong to the object specified by hAny.

Parameters
- hAny: Any kind of handle
- mode: SX_DHCP_OFF: Do not use DHCP
  SX_DHCP_ON: Use DHCP

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetDHCP( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int *mode );
ERR GetEthernetDHCP( HNDL hAny, ref int mode );

Description
Queries the DHCP client setting of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit’s settings.

Parameters
- hAny: Any kind of handle
- mode: Where to store the mode
  SX_DHCP_OFF: Do not use DHCP
  SX_DHCP_ON: Use DHCP

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set IP address

int SxSetEthernetIP( SX_HNDL_UNIT hUnit, char *adrs );
ERR SetEthernetIP( HNDL hUnit, string adrs );

Description
Sets the Ethernet IP address of the unit specified by hUnit.

Parameters
- hUnit: A unit handle
- adrs: An IP address
  Example: "192.168.21.3"

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetIP( SX_HNDL_UNIT hUnit, char *adrs );
ERR GetEthernetIP( HNDL hUnit, ref string adrs );

Description
Queries the Ethernet IP address of the unit specified by hAny.

Parameters
- hUnit: A unit handle
- adrs: Where to store the IP address

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Subnet Mask
int SxSetEthernetNetMask( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs );
ERR SetEthernetNetMask( HNDL hAny, string adrs );
Description
Sets the subnet of the units that belong to the object specified by hAny.
Parameters
hAny Any kind of handle
adrs A subnet mask
Example: "255.255.255.0"
Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetNetMask( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs );
ERR GetEthernetNetMask( HNDL hAny, ref string adrs );
Description
Queries the subnet mask of the unit specified by hAny. If hAny is a unit group, the function queries the master unit’s settings.
Parameters
hAny Any kind of handle
adrs Where to store the subnet mask
Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Default Gateway
int SxSetEthernetGateway( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs );
ERR SetEthernetGateway( HNDL hAny, string adrs );
Description
Sets the default gateway address of the units that belong to the object specified by hAny.
Parameters
hAny Any kind of handle
adrs A default gateway address
Example: "192.168.21.254"
Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxGetEthernetGateway( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, char *adrs );
ERR GetEthernetGateway( HNDL hAny, ref string adrs );
Description
Queries the default gateway address of the unit specified by hAny. If hAny is a unit group, the function queries the master unit’s settings.
Parameters
hAny Any kind of handle
adrs Where to store the default gateway address
Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Query or Set SNTP

```c
int SxSetSNTP( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode, char *serverAddress, char *timeZone, int interval );
```

**Description**

Sets the SNTP client of the units that belong to the object specified by `hAny`. If you execute this function with `mode=1`, the time will be set when the function is executed. After that, the time will be set at the interval specified in hours by the `interval` parameter.

**Parameters**

- **hAny**: Any kind of handle
- **mode**:  
  - `SX_SNTP_DISABLE`: Do not use SNTP
  - `SX_SNTP_ENABLE`: Use SNTP
- **serverAddress**: The SNTP server address
  - Example: "192.168.21.1"
- **timeZone**: The time difference from UTC.
  - Format: "hh:mm" or "–hh:mm"
  - Example: In Japan, the time difference would be "09:00"
- **interval**: The interval at which to set the time (specified in hours)
  - You can set the interval to 0, 1, 2, 4, 6, 8, 12, or 24 hours.
  - If you specify 0, the time will not be set at regular intervals.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if it does not.

```c
int SxGetSNTP( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int *mode, char *serverAddress, char *timeZone, int *interval );
```

**Description**

Queries the SNTP client setting of the unit that belongs to the object specified by `hAny`. If `hAny` is a unit group, the function queries the master unit’s settings. Allocate at least 16 bytes of memory area for `serverAddress`.

**Parameters**

- **hAny**: Any kind of handle
- **mode**: Where to store the mode
  - `SX_SNTP_DISABLE`: Do not use SNTP
  - `SX_SNTP_ENABLE`: Use SNTP
- **serverAddress**: Where to store the SNTP server address
- **timeZone**: The time difference from UTC.
  - Format: "hh:mm" or "–hh:mm"
  - Example: In Japan, the time would be "09:00"
- **interval**: Where to store the interval at which to set the time (specified in hours)
  - Zero indicates that the time will not be set at regular intervals.

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Query or Set Key Lock

### Function Details

**int SxSetKeyLock( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode );**

**ERR SetKeyLock( HNDL hAny, BOOL mode );**

**Description**

Sets the key lock setting of the units that belong to the object specified by hAny.

**Parameters**

- **hAny**: Any kind of handle
- **mode**: SX_KEYLOCK_OFF: Do not lock keys
  SX_KEYLOCK_ON: Lock keys

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Function Details

**int SxGetKeyLock( SX_HNDL_GROUP | SX_HNDL_UNIT hAny, int mode );**

**ERR GetKeyLock( HNDL hAny, ref BOOL mode );**

**Description**

Queries the key lock setting of the unit specified by hAny.
If hAny is a unit group, the function queries the master unit’s settings.

**Parameters**

- **hAny**: Any kind of handle
- **mode**: Where to store the mode
  SX_KEYLOCK_OFF: Do not lock keys
  SX_KEYLOCK_ON: Lock keys

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Get Error Code

**int SxGetError( SX_HNDL_UNIT hUnit, int *error );**

**ERR GetError( HNDL hAny, ref int error );**

**Description**

Retrieves an error code from the error cue of the unit specified by hUnit. If hUnit is a unit group, the error code is retrieved from the master unit.
If the error cue is empty, the function returns zero.

**Parameters**

- **hUnit**: A unit handle
- **error**: Where to store the error code
  Zero: The error cue is empty
  Nonzero: An error code

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.17 Internal Media Operations

Query Drive Count

int SxFileGetDriveNum( SX_HNDL_UNIT hUnit, int *driveNum );

ERR FileGetDriveNum( HNDL hUnit, ref int driveNum );

Description
Retrieves the number of internal media drives of the unit specified by hUnit.

Parameters
- hUnit: A unit handle
- driveNum: Where to store the number of drives

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query Drive Information

int SxFileGetDriveInfo( SX_HNDL_UNIT hUnit, int index, char *driveName, int *totalSize, int *freeSize );

ERR FileGetDriveInfo( HNDL hUnit, int index, ref string driveName, ref int totalSize, ref int freeSize );

Description
Retrieves the drive information of the drive with the specified index in the unit specified by hUnit.
Allocate at least 8 bytes of memory area for driveName.

Parameters
- hUnit: A unit handle
- index: A drive number (starting with 0)
- driveName: Where to store the drive name
  Example: "HD-0"
- totalSize: Where to store the total size (in kilobytes)
- freeSize: Where to store the available memory (in kilobytes)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Current Drive

int SxFileSetCurrentDrive( SX_HNDL_UNIT hUnit, char *driveName );

ERR FileSetCurrentDrive( HNDL hUnit, string driveName );

Description
Sets the current drive (the target drive) of the unit specified by hUnit to the drive specified by driveName.

Parameters
- hUnit: A unit handle
- driveName: A drive name
  Example: "HD-0"

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
int SxFileGetCurrentDrive( SX_HNDL_UNIT hUnit, char *driveName );

ERR FileGetCurrentDrive( HNDL hUnit, ref string driveName );

Description
Retrieves the current drive (the target drive) of the unit specified by hUnit. Allocate at least 8 bytes of memory area for driveName.

Parameters
- hUnit: A unit handle
- driveName: Where to store the drive name (requires 8 or more bytes of memory)

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Query or Set Current Directory
int SxFileChDir( SX_HNDL_UNIT hUnit, char *pathName );

ERR FileChDir( HNDL hUnit, string pathName );

Description
Set the current directory (the target directory) of the unit specified by hUnit to the directory specified by pathName.

Parameters
- hUnit: A unit handle
- pathName: A directory path

Example:

"/abcdefg/efghi" > An absolute path specification
"ijklmn" > A relative path specification
".." > To a new directory
"/" > To the root directory

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

int SxFileCwDir( SX_HNDL_UNIT hUnit, char *pathName );

ERR FileCwDir( HNDL hUnit, ref string pathName );

Description
Retrieves the absolute path name of the current directory (the target directory) of the unit specified by hUnit. Allocate at least 256 bytes of memory area for pathName.

Parameters
- hUnit: A unit handle
- pathName: Where to store the path (requires 256 or more bytes of memory)

Example: "/abcdefg/efghi"

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Create Subdirectory

```c
int SxFileMkDir( SX_HNDL_UNIT hUnit, char *dirName );
```

**ERR FileMkDir( HNDL hUnit, string dirName );**

**Description**
Creates a subdirectory with the name specified by dirName in the current directory of the unit specified by hUnit.

**Parameters**
- **hUnit** A unit handle
- **dirName** The name of the subdirectory
  Example: "jklmn"

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete Subdirectory

```c
int SxFileRmDir( SX_HNDL_UNIT hUnit, char *dirName );
```

**ERR FileRmDir( HNDL hUnit, string dirName );**

**Description**
Deletes the subdirectory with the name specified by dirName in the current directory of the unit specified by hUnit. All of the files in the directory are also deleted.

**Parameters**
- **hUnit** A unit handle
- **dirName** A subdirectory name
  Example: "jklmn"

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Get File Number

```c
int SxFileGetFileNum( SX_HNDL_UNIT hUnit, int *fileNum );
```

**ERR FileGetFileNum( HNDL hUnit, ref int fileNum );**

**Description**
Retrieves the number of files (including subdirectories) in the current directory of the unit specified by hUnit. Use this function before getting file information with SxFileGetFileInfo().

**Parameters**
- **hUnit** A unit handle
- **dirName** A subdirectory name
  Example: "jklmn"

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Get File Information

```
int SxFileGetFileInfo(SX_HNDL_UNIT hUnit, int index, char *fileName, int *attr, int *size, char *
*datetime);
```

**Description**
Retrieves the information of the file specified by index in the current directory of the unit specified by hUnit. Allocate at least 256 bytes of memory area for fileName. Allocate at least 17 bytes of memory area for datetime.
Before using this function, execute SxFileGetFileNum() after setting the appropriate current directory.

**Parameters**
- **hUnit**: A unit handle
- **index**: A file number (starting with 0)
- **fileName**: Where to store the file name (requires 256 or more bytes of memory)
  Example: "stuvw.xyz"
- **attr**: Where to store the file attributes
  The attributes are expressed using the logical sum of the following values.
  - SX_ATTR_SUBDIR: Subdirectory
    (Currently, only the Subdirectory attribute listed above is defined.)
- **size**: Where to store the file size (in bytes)
- **datetime**: Where to store the file date and time (requires 17 or more bytes of memory)
  Format: "YYYY/MM/DD-hh:mm" (16 characters + 1 null character)

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Delete File

```
int SxFileDelete(SX_HNDL_UNIT hUnit, char *fileName);
```

**Description**
Deletes the file with the name specified by fileName in the current directory of the unit specified by hUnit. You cannot delete subdirectories with this function. To delete a subdirectories, use SxFileRmDir().

**Parameters**
- **hUnit**: A unit handle
- **fileName**: A file name
  Example: "stuvw.xyz"

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
### Get File

```c
int SxFileGet(SX_HNDL_UNIT hUnit, char *srcFileName, char *dstFileName);
int SxFileGetM(SX_HNDL_UNIT hUnit, char *srcFileName, char *buf, int bsize, int *size);
```

**ERR FileGet( HNDL hUnit, string srcFileName, string dstFileName );**

**ERR FileGetM( HNDL hUnit, string srcFileName, ref any[] buf, int bsize, ref int size );**

**ERR FileGetM( HNDL hUnit, string srcFileName, ref any[] buf, ref int size )**

**Description**

Retrieves the file with the name specified by `srcFileName` in the current directory of the unit specified by `hUnit`.

*SxFileGet()* copies the contents of the retrieved file to the local file specified by `dstFileName`.

*SxFileGetM()* copies the contents of the retrieved file to the memory address specified by `buf`.

**Parameters**

- **hUnit**: A unit handle
- **srcFileName**: The name of the copy source file (on the unit)
  - Example: "stuvw.xyz"
- **dstFileName**: The name of the copy destination file (on the PC)
  - Example: "C:\SL1000\stuvw.xyz"
- **buf**: Where to store the file image
- **bsize**: The buffer size (in bytes) of where the file image will be stored.
- **size**: Where to store the actual size of the file image (in bytes)

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

### Create File

```c
int SxFilePut(SX_HNDL_UNIT hUnit, char *dstFileName, char *srcFileName);
int SxFilePutM(SX_HNDL_UNIT hUnit, char *dstFileName, char *buf, int size);
```

**ERR FilePut( HNDL hUnit, string dstFileName, string srcFileName );**

**ERR FilePutM( HNDL hUnit, string dstFileName, ref any[] buf, int size );**

**ERR FilePutM( HNDL hUnit, string dstFileName, ref any[] buf);**

**Description**

Creates a file with the name specified by `dstFileName` in the current directory of the unit specified by `hUnit`.

*SxFilePut()* copies the contents of the local file specified by `srcFileName`.

*SxFilePutM()* copies the contents of the memory address specified by `buf`.

**Parameters**

- **hUnit**: A unit handle
- **dstFileName**: The name of the file to be created (on the unit)
  - Example: "stuvw.xyz"
- **srcFileName**: The name of the copy source file (from the PC)
  - Example: "C:\SL1000\stuvw.xyz"
- **buf**: Where to store the file image
- **size**: The file image size in bytes

**Return Value**

Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
Format Internal Hard Disk

`int SxFormatHDD(SX_HNDL_GROUP | SX_HNDL_UNIT hAny);`

*ERR FormatHDD( HNDL hAny );*

**Description**
Formats the internal hard disk of the unit that belongs to an object specified by hAny.

**Parameters**
- **hAny** Any kind of handle

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.
3.18 Debugging

Query or Set Trace Mode
int SxTraceSetMode(int mode, const char *fileName);

ERR TraceSetMode( int mode, string fileName );
ERR TraceSetMode( int mode );

Description
Sets the trace mode to the mode specified by the mode parameter. The fileName parameter specifies the file to write to (a local PC file) if file output is set to on.
The application program can execute this function at any time.

Parameters
mode The trace mode
Specified using a logical sum. For information about the values that can be set, see section 3.19.
fileName The file to write to when file output is set to on.
If the value is null, the function will write to the default file name (".\SxAPILog.txt").

Return Value
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

Explanation
The following two methods are also available for changing the trace mode.

With environment variables

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXAPI_TRACE_MODE</td>
<td>The trace mode (example: 0x1131)</td>
</tr>
<tr>
<td>SXAPI_TRACE_FILE</td>
<td>Trace file name (example: C:\TEMP\SxAPI.log)</td>
</tr>
</tbody>
</table>

With command line parameters

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-SXAPI_TRACE_MODE = &quot;&lt;the trace mode&gt;&quot;</td>
<td>-SXAPI_TRACE_MODE=0x1131</td>
</tr>
<tr>
<td>-SXAPI_TRACE_FILE = &quot;&lt;the trace file name&gt;&quot;</td>
<td>-SXAPI_TRACE_MODE=C:\TEMP\SxAPI.log</td>
</tr>
</tbody>
</table>

The order of preference for the settings is: those settings made by this function, then command line parameters, then environment variables.
The trace mode default value is zero (no tracing), and the default trace file name is ".\SxTraceLog.txt."
int SxTraceGetMode(int *mode, char *fileName);

**ERR TraceGetMode( ref int mode, ref string fileName );**

**Description**
Retrieves the trace mode and the trace output file name (on the PC).

**Parameters**
- **mode** Where to store the trace mode
  Expressed using a logical sum. For information about the values that can be set, see section 3.19.
- **fileName** Where to store the name of the file that is written to when file output is set to on.

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**Output Trace**

int SxTracePrint( char *s );
int SxTracePrintf( char *fmt, ... );

**ERR TracePrint( string s );**

**Description**
SxTracePrint() outputs the string specified by the s parameter to the trace log.
SxTracePrintf() outputs the specified string using the same format as printf().
This function can only be used with VC++.

**Parameters**
- **s** A string (ending with a null character)
- **fmt** A string in printf() format

**Return Value**
Returns zero if the function succeeds and a nonzero error code if the function does not succeed.

**Reset Performance Timer**

unsigned long SxResetTimer();

**uint ResetTimer();**

**Description**
Executing this function resets the performance timer to 0. The function returns the timer value (in milliseconds) immediately preceding the reset.

**Parameters**
None

**Return Value**
- **unsigned long** The timer value immediately preceding the reset (in microseconds)

**Explanation**
The performance timer is reset to zero when an SxAPI function is first called (this first function is ordinarily SxIint). Afterwards, the timer continues counting until this function is executed.
The performance timer is accurate to the microsecond.
3.18 Debugging

Get Performance Timer
unsigned long SxGetTimer();
uint GetTimer();

Description
This function returns the current performance timer value (in microseconds) when it is executed.

Parameters
None

Return Value
unsigned long The timer value immediately preceding the reset (in microseconds)
3.19 Definitions

Handles

typedef unsigned long SX_HNDL; // Any type of handle
#define SX_HNDL_COMM SX_HNDL // Communication handle
#define SX_HNDL_GROUP SX_HNDL // Unit group handle
#define SX_HNDL_UNIT SX_HNDL // Unit handle
#define SX_HNDL_MOD SX_HNDL // Module handle
#define SX_HNDL_CH SX_HNDL // Channel handle
#define SX_HNDL_MEASGRP SX_HNDL // A measuring group handle

Communication Types

#define SX_WIRE_USB 7 // USB (USBTMC)
#define SX_WIRE_LAN 8 // Ethernet (VXI-11)

Module Information Structure

typedef struct {
    int slotNo; // Slot number
    int type; // 0: empty, 1: AcqModule
    int measGroupNo; // A measuring group number (0 to 3)
    int chOffsetGrp; // Internal unit group channel offset number
    int chOffsetUnit; // Internal unit channel offset number
    int chNum; // Number of channels
    char productCode[16]; // Module code (example: "M701250")
    char productName[16]; // Module name (example: "HS10M12")
} SX_INFO_MOD;

Unit Information Structure

typedef struct {
    int groupID; // Unit group number
    int unitID; // Unit number
    char address[64]; // Unit address
    int chOffsetGrp; // Internal unit group channel offset number
    int modNum; // Number of modules
    char productCode[16]; // Product model (example: "720120")
    char groupName[32]; // Unit name (example: "Unit1")
    char unitName[32]; // Options (example: "128MW,HD,ETHER")
    SX_INFO_MOD moduleInfo[10]; // Module information
} SX_INFO_UNIT;

Unit Group Information Structure

typedef struct {
    int groupID; // Unit group number
    int errorCode; // Unit group error code (can be opened if the code is
                    // 0). For details, see “Unit Group Error Codes.”
    int unitNum; // Number of units
    char groupName[32]; // When errorCode!=0 the value is “” (empty)
} SX_INFO_GROUP;
3.19 Definitions

Acquisition Data Information Structure

typedef struct {
    INT channel; // Logic channel number (starting with 0)
    UCHAR dataType; // Raw data type
    CHAR startBit; // Location of the first effective bit (starting with 0) in an integer.
    CHAR effectiveBit; // Effective bit length of an integer (0 is valid until lsb)
    char pad1[1] // Label name
    char label[8]
    double vResolu; // Physical value conversion gain
    double vOffset; // Physical value conversion offset
    UNION64 nonData; // Non-display code (raw data or something close)
    char unit[4]; // Unit of measurement string
    INT recordLen; // Measurement points (data points)
    INT trigPos; // Trigger position (Only valid in Triggered mode. The value is 0x80000000 when it is invalid.)
    char pad3[8];
    char pad4[8];
    char pad5[8];
} SX_INFO_CH;

Event Types

#define SX_EV_ALA 0x00000080 // System alarm has occurred.
#define SX_EV_ACQ_START 0x00010000 // Measurement has started.
#define SX_EV_ACQ_STOP 0x00020000 // Measurement has ended.
#define SX_EV_TRIG_START 0x00040000 // A trigger has been detected (in Triggered mode).
#define SX_EV_TRIG_END 0x00080000 // A single triggered measurement has ended.
#define SX_EV_REC_START 0x01000000 // Recording has started.
#define SX_EV_SAVE_SATRT 0x02000000 // The SL1000 has started saving waveform data.
#define SX_EV_REC_END 0x04000000 // Recording has ended.
#define SX_EV_ACQ_DATA_READY 0x08000000 // The specified number of data points have been acquired (in Free Run mode).
#define SX_EV_SAVE_END 0x10000000 // The SL1000 has finished saving waveform data
#define SX_EV_CHANNEL_ALARM 0x20000000 // A channel alarm has occurred or been released
Error Codes

#define SX_ERR_OK 0 // No error, closed properly
#define SX_ERR_TIMEOUT 10001 // Timeout
#define SX_ERR_NO_STATION 10002 // Cannot find the target unit
#define SX_ERR_FAIL_OPEN 10003 // Open failed
#define SX_ERR_NOT_OPEN 10004 // Not opened
#define SX_ERR_ALREADY_OPEN 10005 // Already opened
#define SX_ERR_NOT_CONTROL 10006 // Environment error
#define SX_ERR_ILLEGAL_PARAMETER 10007 // Invalid parameter
#define SX_ERR_SEND_ERROR 10008 // Send error
#define SX_ERR_RECV_ERROR 10009 // Receive error
#define SX_ERR_NOT_BLOCK 10010 // The received data is not in block format.
#define SX_ERR_SYSTEM_ERROR 10011 // System error
#define SX_ERR_ILLEGAL_ID 10012 // ID violation
#define SX_ERR_COMM_ERROR 10013 // Communication command error
#define SX_ERR_BUFFER_SHORT 10014 // Insufficient buffer
#define SX_ERR_NO_GROUP 10016 // Cannot find the target unit group
#define SX_ERR_ILLEGAL_ID 10017 // Invalid unit group
#define SX_ERR_HNDL_TYPE 10018 // Handle type violation
#define SX_ERR_ILLEGAL_HNDL 10019 // Handle error
#define SX_ERR_NO_HNDL 10020 // Cannot find handle
#define SX_ERR_ILLEGAL_MESSAGE 10021 // Command string violation
#define SX_ERR_OUT_OF_RANG 10022 // Data outside of the range has been specified.
#define SX_ERR_NO_DATA 10023 // The specified data does not exist.
#define SX_ERR_CONFLICT 10024 // Conflict error
#define SX_ERR_INTERNAL_ERROR 10031 // Internal error

Unit Group Error Codes

#define SX_GRPERR_OK 0 // Group can be opened
#define SX_GRPERR_NOUNIT 1 // There are no units.
#define SX_GRPERR_MISSING 2 // The unit ID is missing. (The problem ID is b8 to b15).
#define SX_GRPERR_DUPLICATE 3 // There are duplicate IDs. (The problem ID is b8 to b15.)
#define SX_GRPERR_SYNCLINE 4 // Sync cable not connected. (The problem ID is b8 to b15.)

Self Test Results

#define SX_SELFTEST_ACQMEM 0x0001 // Waveform memory error
#define SX_SELFTEST_SYSMEM 0x0002 // System memory error
#define SX_SELFTEST_BACKUPMEM 0x0004 // Backup memory error
#define SX_SELFTEST_HDD 0x0010 // Internal hard disk error

File Attribute

#define SX_ATTR_SUBDIR 0x0001 // Subdirectory
### Trace Modes

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#define SX_TRACE_FILE</code></td>
<td>0x00000001</td>
<td>File output on</td>
</tr>
<tr>
<td><code>#define SX_TRACE_DEBUGOUT</code></td>
<td>0x00000002</td>
<td>Debug output on (Only the debugging DLL is active. <code>SX_TRACE_FILE</code> has priority.)</td>
</tr>
<tr>
<td><code>#define SX_TRACE_SXAPI</code></td>
<td>0x00000010</td>
<td>SxAPI.dll access log</td>
</tr>
<tr>
<td><code>#define SX_TRACE_SRQ</code></td>
<td>0x00000020</td>
<td>SRQ processing log</td>
</tr>
<tr>
<td><code>#define SX_TRACE_TMCTL</code></td>
<td>0x00000040</td>
<td>tmctl.dll access log</td>
</tr>
<tr>
<td><code>#define SX_TRACE_TIME</code></td>
<td>0x00000100</td>
<td>Timestamp</td>
</tr>
<tr>
<td><code>#define SX_TRACE_ELASP</code></td>
<td>0x00000200</td>
<td>Relative timestamp</td>
</tr>
<tr>
<td><code>#define SX_TRACE_PERFORM</code></td>
<td>0x00000400</td>
<td>Performance measurement</td>
</tr>
<tr>
<td><code>#define SX_TRACE_ONLY_CALL</code></td>
<td>0x00001000</td>
<td>Only trace function calls</td>
</tr>
<tr>
<td><code>#define SX_TRACE_ONLY_RETURN</code></td>
<td>0x00002000</td>
<td>Only trace function returns</td>
</tr>
<tr>
<td><code>#define SX_TRACE_AUTOFILENAME</code></td>
<td>0x00010000</td>
<td>Automatically insert the date, time, and a number after the file name (only valid when <code>SX_TRACE_FILE</code> is set to on)</td>
</tr>
</tbody>
</table>
4.1 Using Communication Commands

Functions for Sending and Receiving Communication Commands

Communication commands can be sent and received using the following SxAPI communication command control functions.

- `SxSetControl()` Sends a setup command
- `SxSetControlBinary()` Sends a setup command with the parameters in binary format
- `SxGetControl()` Sends and receives query commands
- `SxGetControlBinary()` Send and receives query commands with the parameters in binary format

For details about communication command control functions, see chapter 3.8.

Syntax Rules

- Parts of commands and parameters that are written in lowercase can be omitted.
  Example: `ALAR:ACK:EXEC`
- Units do not distinguish between uppercase and lowercase letters in commands and parameters.
  Example: `alar:ack:exec`
- Parts of commands and parameters surrounded by square brackets ([]) can be omitted.
  Example: `:TRIG:HYST MIDD`
- Parts of commands and parameters surrounded by curly brackets ({}) are variables. They cannot be omitted.
  Example: `CHAN<ch>:LSC:PL1X 10`
- Vertical lines ( | ) indicate that a choice must be made from one of the items that they separate.
  Example: `:ALAR:COMB OR`
- `<boolean>` indicates a boolean type. Boolean types are switched to true by 1 or on, and switched to false by 0 or off.
  Example: `GONO:ACT:BUZZ OFF`
- `<NRf>` indicates a numeric value.
  Example: `:TRIG:LEV 2.5`
- `<Character string data>` indicates a string surrounded by double quotation marks.
  Example: `ALAR:ACT:MAIL:ADDR "yoko@yokogawa.jp.com"`
- `<Block data>` indicates a binary format parameter.
- `<ch>` indicates a channel number. SxAPI will automatically produce the appropriate text, so you can just enter “<ch>” as-is.
  Example: `CHAN<ch>:LSC:PL1X 10`
- `<mo>` indicates a module number. SxAPI will automatically produce the appropriate text, so you can just enter “mo” as-is.
- `<sg>` indicates a measuring group number. SxAPI will automatically produce the appropriate text, so you can just enter “sg” as-is.
- Other words surrounded by <> are indicative of the kinds of values that should be used in their place.

Note

Do not use commands that are not listed here (even if they are listed in IEEE 488.2-1987).
Doing so may cause SxAPI to malfunction.
# 4.2 Commands

## ALARm Group

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ALARm:ACK:EXECute</td>
<td>Releases alarm output</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:ACK:EXECute?</td>
<td>Queries the acquisition number, which is counted from the start of measurement, when an alarm occurs.</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:COMBination</td>
<td>Sets or queries the AND/OR state of the alarms of each channel.</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:CONDition?</td>
<td>Queries the alarm output terminal condition.</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:CONDition?</td>
<td>Queries the alarm condition of the specified channel.</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:HYSteresis&lt;X2&gt;</td>
<td>Sets or queries the alarm hysteresis of a channel using three levels.</td>
<td>4-6</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:NHYSteresis&lt;X2&gt;</td>
<td>Sets or queries the alarm hysteresis of a channel using numeric values.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:LEVe1&lt;X2&gt;</td>
<td>Sets or queries the alarm level of a channel (when the input of the specified channel is not logic).</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:TYPE</td>
<td>Sets or queries the alarm type of a channel.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CHANnel1&lt;ch&gt;:AVAlue?</td>
<td>Queries the measured value at the alarm occurrence on the specified channels as an ASCII string.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CMODe</td>
<td>Sets or queries alarm hold.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CONDition?</td>
<td>Sets or queries alarm operation mode.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:CMODe</td>
<td>Sets or queries the channel alarm operation mode.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:SMODe</td>
<td>Sets or queries the system alarm operation mode.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:OTErmina1</td>
<td>Sets or queries the alarm output terminal on/off state.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:SOURce</td>
<td>Sets or queries the alarm detection source.</td>
<td>4-7</td>
</tr>
<tr>
<td>:ALARm:STATus?</td>
<td>Queries the channel alarm status.</td>
<td>4-8</td>
</tr>
<tr>
<td>:ALARm:SSTATus?</td>
<td>Queries the system alarm status value.</td>
<td>4-8</td>
</tr>
<tr>
<td>:ALARm:SYSTem:SOURce:BOVerrun</td>
<td>Sets or queries the system alarm buffer overrun detection.</td>
<td>4-8</td>
</tr>
<tr>
<td>:ALARm:SYSTem:SOURce:FSTop</td>
<td>Sets or queries the system alarm fan stop detection.</td>
<td>4-8</td>
</tr>
<tr>
<td>:ALARm:SYSTem:SOURce:DFULl</td>
<td>Sets or queries the system alarm HDD full detection.</td>
<td>4-8</td>
</tr>
<tr>
<td>:ALARm:STIMe?</td>
<td>Queries the time of the most recent system alarm condition change.</td>
<td>4-8</td>
</tr>
</tbody>
</table>

## CHANnel Group

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:BIASt</td>
<td>Sets or queries the on/off state of the acceleration sensor’s bias current (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:BWIDt</td>
<td>Sets or queries the filter (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:COUPling</td>
<td>Sets or queries input coupling (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:GAIN</td>
<td>Sets or queries the gain (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:SENSitivity</td>
<td>Sets or queries the sensitivity (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:ACCL:UNIT</td>
<td>Sets or queries the unit of measurement of the upper and lower limit values (Acceleration/Voltage Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:COLor</td>
<td>Sets the channel waveform color or queries the current setting.</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:BWIDt</td>
<td>Sets or queries the bandwidth limit (Frequency Module).</td>
<td>4-9</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:CELimination</td>
<td>Sets or queries chattering elimination (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:COUPling</td>
<td>Sets or queries input coupling (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:HYSteresis</td>
<td>Sets or queries hysteresis (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:PRESet</td>
<td>Sets or queries the preset (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:PROBe</td>
<td>Sets or queries the probe attenuation (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:FULLup</td>
<td>Sets or queries the pull-up on/off state (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:SLOPe</td>
<td>Sets or queries the slope (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:THReshold</td>
<td>Sets or queries the threshold level (Frequency Module).</td>
<td>4-10</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:INPut:VRANGE</td>
<td>Sets or queries the voltage range (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LSCale:AVAlue</td>
<td>Sets or queries linear scaling coefficient A (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LSCale:BVAlue</td>
<td>Sets or queries linear scaling coefficient B (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LSCale:GETMeasure</td>
<td>Measures the X values of P1 and P2 for linear scaling (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LSCale:MODE</td>
<td>Sets or queries linear scaling (Frequency Module).</td>
<td>4-11</td>
</tr>
</tbody>
</table>
### 4.2 Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LScale:[:P1X</td>
<td>P1Y</td>
<td>P2X</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:LScale:UNIT</td>
<td>Sets or queries the unit of measurement to attach to the result of linear scaling (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:OFFSET</td>
<td>Sets or queries the offset value (Frequency Module).</td>
<td>4-11</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:CFrequency</td>
<td>Sets or queries the center frequency (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:DECLEration</td>
<td>Sets or queries the on/off state of deceleration prediction (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:DPULse</td>
<td>Sets or queries the distance per pulse (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:FiLTer:SMoothing:MODE</td>
<td>Sets or queries the on/off state of smoothing (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:FiLTer:SMoothing:VALUE</td>
<td>Sets or queries the moving average order of smoothing (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:FiLTer:PAVerage:MODE</td>
<td>Sets or queries the on/off state of pulse average (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:FiLTer:PAVerage:VALUE</td>
<td>Sets or queries the pulse average count (Frequency Module).</td>
<td>4-12</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:FiLTer:FUNCTION</td>
<td>Sets or queries the measuring mode (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:LRSet</td>
<td>Sets or queries the over limit reset (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:MPULse</td>
<td>Sets or queries whether the measurement pulse is positive or negative (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:PROTate</td>
<td>Sets or queries the number of pulses per rotation (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:RESet</td>
<td>Resets the pulse count (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:STOpPredict</td>
<td>Sets or queries the on/off state of stop prediction (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:TUNIT</td>
<td>Sets or queries the time unit (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:UNIT</td>
<td>Sets or queries the pulse integration unit (Frequency Module).</td>
<td>4-13</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:SetUp:UPULse</td>
<td>Sets or queries the unit/pulse (Frequency Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:FREQ:VDIV</td>
<td>Sets or queries the Value/Div (Frequency Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:BA lance:EXE C</td>
<td>Sets or queries the channel on which balancing is to be executed (Strain Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:BA lance:EXE C:EXECut e</td>
<td>Balances strain (Strain Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:EXCitation</td>
<td>Sets or queries the filter (Strain Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:GFACtor</td>
<td>Sets or queries the gauge factor (Strain Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:INVert</td>
<td>Sets or queries whether the display is inverted (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:AV ALue</td>
<td>Sets or queries linear scaling coefficient A (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:BVA Lue</td>
<td>Sets or queries offset value B (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:BALance:DEC</td>
<td>Sets or queries the display format for linear scaling (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:DEC</td>
<td>Sets or queries the channel on which balancing is to be executed (Strain Module).</td>
<td>4-14</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:DEC:EXECute</td>
<td>Sets or queries the decimal place when the display format for linear scaling is set to Floating (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:MODE</td>
<td>Sets or queries the linear scaling method (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:[:P1X</td>
<td>P1Y</td>
<td>P2X</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:SHUNT</td>
<td>Executes shunt calibration (Strain Module with DSUB, Shunt-Cal).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:LScale:UNIT</td>
<td>Sets or queries the unit of measurement that is attached to the result of linear scaling (Strain Module).</td>
<td>4-15</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:RANGE</td>
<td>Sets or queries the measuring range (Strain Module).</td>
<td>4-16</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:STRain:UNIT</td>
<td>Sets or queries the unit of measurement (Strain Module).</td>
<td>4-16</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:BURNOut</td>
<td>Sets or queries whether or not burnout is detected (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-16</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:BWIDTH</td>
<td>Sets or queries the bandwidth limit (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-16</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:COUPLing</td>
<td>Sets or queries input coupling (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-16</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:RJ C</td>
<td>Sets or queries the RJC (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-17</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:TYPE</td>
<td>Sets or queries the thermocouple type (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-17</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;:TEMPerature:UNIT</td>
<td>Sets or queries the unit of measurement values (Temperature, High Precision Voltage Isolation Module).</td>
<td>4-17</td>
</tr>
</tbody>
</table>
## 4.2 Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:BWIDth</td>
<td>Sets or queries the bandwidth limit (voltage modules)*.</td>
<td>4-17</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:COUPling</td>
<td>Sets or queries input coupling (voltage modules)*.</td>
<td>4-17</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:INVert</td>
<td>Sets or queries whether or not the display is inverted (voltage modules)*.</td>
<td>4-17</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale: BVALue</td>
<td>Sets or queries linear scaling offset value B (voltage modules).</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:DISPLAYtype:MODE</td>
<td>Sets or queries the display format for linear scaling.</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:DISPLAYtype:DECimalnum</td>
<td>Sets or queries decimal place when the display format for linear scaling is set to Floating.</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:DISPLAYtype:SUBunit</td>
<td>Sets or queries the sub unit when the display format for linear scaling is set to Floating.</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:GetMeasure</td>
<td>Measures the X values of P1 and P2 for linear scaling (voltage modules)*.</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:MODE</td>
<td>Sets or queries linear scaling (voltage modules)*.</td>
<td>4-18</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:[FIX</td>
<td>PLY</td>
<td>PZX</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:LScale:UNIT</td>
<td>Sets or queries the unit of measurement to attach to the result of linear scaling (voltage modules)*.</td>
<td>4-19</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:PROBe</td>
<td>Sets or queries the probe type (voltage modules)*.</td>
<td>4-19</td>
</tr>
<tr>
<td>:CHANnel&lt;ch&gt;[:VOLTage]:VDIV</td>
<td>Sets or queries the V/div value (voltage modules).</td>
<td>4-19</td>
</tr>
</tbody>
</table>

* Voltage module refers to the High-Speed 100 MS/s, 12-Bit Isolation Module; the High-Speed 10 MS/s, 12-Bit Isolation Module; the High-Speed High-Resolution 1 MS/s, 16-Bit Isolation Module; the High-Speed 10 MS/s, 12-Bit Non-Isolation Module; and the High-Voltage 100 kS/s, 16-Bit Isolation Module (with RMS).

### GONogo Group

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:GONogo:ACONDition</td>
<td>Sets or queries the GO/NO-GO judgment action condition.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:BUZZer</td>
<td>Sets or queries whether or not a beep is sounded when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:MAIL:ADDRESS</td>
<td>Sets or queries the destination e-mail address for when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:MAIL:COUNT</td>
<td>Sets or queries the e-mail transmission limit for when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:MAIL:MODE</td>
<td>Sets or queries whether or not an e-mail is sent when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:SAVE[:MODE]</td>
<td>Sets or queries whether or not waveform data is saved to the storage media when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:ACTION:SAVE:TYPE</td>
<td>Sets or queries the data type for saving waveform data to the storage media when the condition is met.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:AREA</td>
<td>Sets or queries the waveform area that is judged.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:COUNt?</td>
<td>Queries the number of performed GO/NO-GO judgments.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:LOGic</td>
<td>Sets or queries the GO/NO-GO logical condition.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:MODE</td>
<td>Sets or queries the GO/NO-GO judgment mode.</td>
<td>4-20</td>
</tr>
<tr>
<td>:GONogo:NCOUNT?</td>
<td>Queries the GO/NO-GO judgment NO-GO count.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::CAUSE?</td>
<td>Queries whether or not the specified waveform parameter is the cause of a NO-GO judgment.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::MODE</td>
<td>Sets or queries whether or not the specified waveform parameter is OFF, or what its judgment criterion is.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::PARAM?</td>
<td>Queries the waveform parameter of the specified judgment condition.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::TRACe</td>
<td>Sets or queries the channel number of the specified judgment condition.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::TYPE: &lt;Parameter&gt;</td>
<td>Sets or queries the upper and lower limits of the judgment area for the specified judgment condition.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:PARAMeter:ITEM&lt;x&gt;::VALue?</td>
<td>Queries the automated measurement value of the specified GO/NO-GO judgment parameter.</td>
<td>4-21</td>
</tr>
<tr>
<td>:GONogo:RStatus?</td>
<td>Queries the most recent GO/NO-GO judgment.</td>
<td>4-21</td>
</tr>
</tbody>
</table>
### MEASure Group

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:MEASure:AREA</td>
<td>Sets or queries the automatically measured waveform area for the waveform parameters.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:DPRoximal:MODE</td>
<td>Sets or queries the distal, mesial, and proximal point mode setting.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:DPRoximal:PERCent</td>
<td>Sets or queries the distal, mesial, and proximal points as percentages.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:DPRoximal:UNIT</td>
<td>Sets or queries the distal, mesial, and proximal points.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:METHOD</td>
<td>Sets or queries the high/low point setting method.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:&lt;Parameter&gt;:STATE</td>
<td>Sets or queries the on/off state of the measurement of the specified waveform parameter.</td>
<td>4-22</td>
</tr>
<tr>
<td>:MEASure:CHANnel&lt;ch&gt;:&lt;Parameter&gt;:VALue?</td>
<td>Queries the automated measurement value of the specified waveform parameter.</td>
<td>4-23</td>
</tr>
<tr>
<td>:MEASure:CRANge</td>
<td>Sets or queries the waveform parameter measurement range.</td>
<td>4-23</td>
</tr>
<tr>
<td>:MEASure:MODE</td>
<td>Sets or queries the waveform parameter automated measuring mode.</td>
<td>4-23</td>
</tr>
<tr>
<td>:MEASure:WAIT?</td>
<td>Waits for the execution of waveform parameter automated measurement with a set timeout.</td>
<td>4-23</td>
</tr>
</tbody>
</table>

### TRIGger Group

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>:TRIGger:COMBination:CHANnel&lt;ch&gt;:COMBination</td>
<td>Sets or queries the trigger AND/OR state of all of the bits in the specified channel for the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:COMBination:CHANnel&lt;ch&gt;:HYSTeresis&lt;x&gt;</td>
<td>Sets or queries the trigger hysteresis of the specified channel in the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:COMBination:CHANnel&lt;ch&gt;:LEVEL&lt;x&gt;</td>
<td>Sets or queries the trigger level of the specified channel in the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:COMBination:CHANnel&lt;ch&gt;:TYPE</td>
<td>Sets or queries the trigger type of the specified channel in the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:COMBination:EXTERNAL:TYPE</td>
<td>Sets or queries the external trigger type of the specified channel in the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:COMBination:MODE</td>
<td>Sets or queries the combination mode of the combination trigger class.</td>
<td>4-24</td>
</tr>
<tr>
<td>:TRIGger:DELay</td>
<td>Sets or queries the delay (the time between the trigger point and the trigger position)</td>
<td>4-25</td>
</tr>
<tr>
<td>:TRIGger:HOLDoff:TIME</td>
<td>Sets or queries the trigger hold off time.</td>
<td>4-25</td>
</tr>
<tr>
<td>:TRIGger[:SIMple]:HYSTeresis</td>
<td>Sets or queries the hysteresis of the simple trigger.</td>
<td>4-25</td>
</tr>
<tr>
<td>:TRIGger[:SIMple]:LEVEL</td>
<td>Sets or queries the simple trigger of the channel specified by :TRIGger[:SIMple]:SOURce.</td>
<td>4-25</td>
</tr>
<tr>
<td>:TRIGger[:SIMple]:SLOPe</td>
<td>Sets or queries the simple trigger type of the channel specified by :TRIGger[:SIMple]:SOURce.</td>
<td>4-26</td>
</tr>
<tr>
<td>:TRIGger:SIMple:SOURce</td>
<td>Sets or queries the simple trigger source.</td>
<td>4-26</td>
</tr>
<tr>
<td>:TRIGger:TIMer:DATE</td>
<td>Sets or queries the date of the time trigger.</td>
<td>4-26</td>
</tr>
<tr>
<td>:TRIGger:TIMer:INTERval</td>
<td>Sets or queries the trigger interval of the time trigger.</td>
<td>4-26</td>
</tr>
<tr>
<td>:TRIGger:TIMer:TIME</td>
<td>Sets or queries the time of the time trigger.</td>
<td>4-26</td>
</tr>
<tr>
<td>:TRIGger:TYPE</td>
<td>Sets or queries the trigger type.</td>
<td>4-26</td>
</tr>
</tbody>
</table>
4.3 ALARm Group

ALARm group commands are only valid when the measuring mode is Free Run mode.

:ALARm:ACK:EXECute
Function: Releases alarm output.
Syntax: :ALARm:ACK:EXECute

:ALARm:ACOunt?
Function: Queries the acquisition number, which is counted from the start of measurement, when an alarm occurs.
Syntax: :ALARm:ACOunt?
Example: :ALARm:ACOunt? -> 65535

:ALARm:COMBination
Function: Sets or queries the AND/OR state of the alarms of each channel.
Syntax: :ALARm:COMBination {AND|OR}
Example: :ALARm:COMBination OR

:ALARm:CONDition?
Function: Queries the alarm output terminal condition.
Syntax: :ALARm:CONDition?
Example: :ALARm:CONDition? -> 1
Description: If output is on, the command returns 1. If output is off, the command returns 0.

:ALARm:CHANnel<ch>:AVALue?
Function: Queries the measured value at the alarm occurrence on the specified channels as an ASCII string.
Syntax: :ALARm:CHANnel<ch>:AVALue?
Description: The measured values of each channel are divided by semicolons (0x3b). The values for all active channels are output. You can use the VERBose setting to decide whether or not returned values have labels and units. If a channel's display is set to Off, its measured value will be listed as “Off.”
Format: 
```
Label, 8 char. | Measured value, 11 char. | Unit, 4 char. (Right justified)
```

:ALARm:CHANnel<ch>:CONDition?
Function: Queries the alarm condition of the specified channel.
Syntax: :ALARm:CHANnel<ch>:CONDition?
Example: :ALARm:CHANnel<ch>:CONDition? -> 1
Description: If the alarm is on, the command returns 1. If the alarm is off, the command returns 0.

:ALARm:CHANnel<ch>:HYSTeresis<X2>
Function: Sets or queries the alarm hysteresis of a channel using three levels.
Syntax: :ALARm:CHANnel<ch>:
HYSTeresis<X2> {HIGH|LOW|MIDDle}
:ALARm:CHANnel<ch>:HYSTeresis<X2>? <X2> = 1, 2
Example: :ALARm:CHANnel<ch>:HYSTeresis1 LOW
Description: If TYPE is HIGH, LOW, only level 1 is used. If TYPE is WLin, WLOut, both level 1 and 2 are used. Level 1 is the upper limit. Level 2 is the lower limit. HIGH: ±10% (of 10 times V/Div) MIDDle: ±5% (of 10 times V/Div) LOW: ±1% (of 10 times V/Div)
### 4.3 ALARm Group

:ALARm:CHANnel<ch>:NHYSteresis<X2>

**Function**: Sets or queries the alarm hysteresis of a channel using numeric values.

**Syntax**:
:ALARm:CHANnel<ch>:NHYSteresis<X2> {<Voltage>|<NRf>|<Current>}

:ALARm:CHANnel<ch>:NHYSteresis<X2>?  
<X2> = 1, 2

- If TYPE is HIGH, LOW, only level 1 is used.
- If TYPE is WLIn, WLOut, both level 1 and 2 are used.

**Example**:
:ALARm:CHANnel<ch>:NHYSteresis2 2.0
:ALARm:CHANnel<ch>:NHYSteresis2?  
-> 2.000E+00

:ALARm:CHANnel<ch>:LEVel<X2>

**Function**: Sets or queries the alarm level of a channel (when the input of the specified channel is not logic).

**Syntax**:
:ALARm:CHANnel<ch>:LEVel<X2> {<Voltage>|<NRf>|<Current>}

:ALARm:CHANnel<ch>:LEVel<X2>?  
<X2> = 1, 2

- If TYPE is HIGH, LOW, only level 1 is used.
- If TYPE is WLIn, WLOut, both level 1 and 2 are used.

**Description**:
Level 1 is the upper limit. Level 2 is the lower limit.

### Example:
:ALARm:CHANnel<ch>:LEVel2 2.0
:ALARm:CHANnel<ch>:LEVel2?  
-> 2.000E+00

:ALARm:CHANnel<ch>:TYPE

**Function**: Sets or queries the alarm type of a channel.

**Syntax**:
:ALARm:CHANnel<ch>:TYPE {HIGH|LOW|OFF|WLIn|WLOut}

:ALARm:CHANnel<ch>:TYPE?

**Description**:
The Au7Fe temperature measuring range is 0 to 280 K (-273 to 7°C)

:ALARm:CMODe

**Function**: Sets or queries the channel alarm operation mode.

**Syntax**:
:ALARm:CMODe {<boolean>}

:ALARm:CMOD?

**Description**:
Sets whether to detect or not detect the channel alarm.

:ALARm:SMODe

**Function**: Sets or queries the system alarm operation mode.

**Syntax**:
:ALARm:SMODe {<boolean>}

:ALARm:SMOD?

**Description**:
Sets whether to detect or not detect the system alarm.

:ALARm:OTERminal

**Function**: Sets or queries the alarm output terminal on/off state.

**Syntax**:
:ALARm:OTERminal {<boolean>}

:ALARm:OTERminal?

**Description**:
If the alarm output terminal is switched off, it will remain off regardless of the alarm settings or conditions.

:ALARm:SOURce

**Function**: Sets or queries the alarm detection source.

**Syntax**:
:ALARm:SOURce {CHANnel|SYSTem}

:ALARm:SOURce?

**Description**:
The initial value is CHANnel.
4.3 ALARm Group

:ALARm:STATUs?
Function Queries the channel alarm status.
Syntax :ALARm:STATUs?
Example :ALARm:STATUs? -> 65535
Description The :ALARm:CHANnel<ch>:CONDition? command must be used repeatedly to acquire the alarm statuses of all channels, so this command returns a bit pattern. The MSB of the returned value (bit 15) represents channel 1. The LSB (bit 0) represents channel 16. A bit value of 1 represents an active alarm status. The channel numbers referred to here are physical channel numbers, which are different from the actual channel numbers.

:ALARm:SSTATUs?
Function Queries the system alarm status value.
Syntax :ALARm:SSTATUs?
Example :ALARm:SSTATUs? -> 2
Description The system status bit assignments are shown in the table below.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>HDD_FULL</td>
<td>The disk is full.</td>
</tr>
<tr>
<td>2</td>
<td>FAN_STOP</td>
<td>The fan has stopped.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BUF_OVERRUN_HOST</td>
<td>Host (PC) buffer has overrun.</td>
</tr>
<tr>
<td>5</td>
<td>BUF_OVERRUN_UNIT</td>
<td>The SL1000 buffer has overrun.</td>
</tr>
<tr>
<td>6</td>
<td>DISCONNECT</td>
<td>The unit was disconnected, or the sync signal between units were lost.</td>
</tr>
<tr>
<td>7 and 8</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>HDD_FULL_GROUP</td>
<td>The hard disk of a unit in the unit group has become full (occurs only on the master unit).</td>
</tr>
<tr>
<td>10 to 13</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DISCONNECT_GROUP</td>
<td>A unit in the unit group was disconnected (occurs only on the master unit).</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

:ALARm:SYSTem:SOURce:BOVerrun
Function Sets or queries the system alarm buffer overrun detection.
Syntax :ALARm:SYSTem:SOURce:BOVerrun {<boolean>}
:ALARm:SYSTem:SOURce:BOVerrun?

:ALARm:SYSTem:SOURce:FSTop
Function Sets or queries the system alarm fan stop detection.
Syntax :ALARm:SYSTem:SOURce:FSTop {<boolean>}
:ALARm:SYSTem:SOURce:FSTop?

:ALARm:SYSTem:SOURce:DFULL
Function Sets or queries the system alarm HDD full detection.
Syntax :ALARm:SYSTem:SOURce:DFULL {<boolean>}
:ALARm:SYSTem:SOURce:DFULL?

:ALARm:STIMe?
Function Queries the time of the most recent system alarm condition change.
Syntax :ALARm:STIMe? -> {<NRf>,<NRf>,<NRf>,<NRf>,<NRf>,<NRf>,<NRf>}
<NRf>: Year (from 2007)
<NRf>: Month (1 to 12)
<NRf>: Day (1 to 31)
<NRf>: Hour (0 to 23)
<NRf>: Minute (0 to 59)
<NRf>: Second (0 to 59)
<NRf>: Millisecond (0 to 900)
Description If no status changes occur after measurement starts, the function will return the time when measurement started.
4.4 CHANnel Group

The CHANnel group functions deal with the vertical axes of the SL1000 channels.

>:CHANnel<ch>:ACCL:BIAS
Function Sets or queries the on/off status of the acceleration sensor’s bias current when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:BIAS {<boolean>}
Example :CHANNEL<ch>:ACCL:BIAS ON
>:CHANnel<ch>:ACCL:BIAS? -> 1
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:ACCL:BWIDTH
Function Sets or queries the filter when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:BWIDTH {FULL|AUTO|<Frequency>}
>:CHANnel<ch>:ACCL:BWIDTH?
<Frequency> = 4 kHz, 400 Hz, 40 Hz
Example :CHANNEL<ch>:ACCL:BWIDTH AUTO
>:CHANNEL<ch>:ACCL:BWIDTH? -> AUTO
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:ACCL:COUPLing
Function Sets or queries input coupling when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:COUPLing {AC|DC|ACCL|GND}
>:CHANnel<ch>:ACCL:COUPLing?
Example :CHANNEL<ch>:ACCL:COUPLING GND
>:CHANNEL<ch>:ACCL:COUPLING? -> GND
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:ACCL:GAIN
Function Sets or queries the gain when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:GAIN {<NRf>}
>:CHANnel<ch>:ACCL:GAIN?
<NRf> = 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100
Example :CHANNEL<ch>:ACCL:GAIN 100
>:CHANNEL<ch>:ACCL:GAIN? -> 100.0
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:ACCL:SENSitivity
Function Sets or queries the sensitivity when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:SENSitivity {<NRf>}
>:CHANnel<ch>:ACCL:SENSitivity?
<NRf> = 0.1 to 2000
Example :CHANNEL<ch>:ACCL:SENSITIVITY 10
>:CHANNEL<ch>:ACCL:SENSITIVITY? -> 10.00
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:ACCL:UNIT
Function Sets or queries the unit of measurement of the upper and lower limit values when an Acceleration/Voltage Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:ACCL:UNIT {<String>}
>:CHANnel<ch>:ACCL:UNIT?
Example :CHANNEL<ch>:ACCL:UNIT "ACCL"
>:CHANNEL<ch>:ACCL:UNIT? -> "ACCL"
Description An error occurs if an Acceleration/Voltage Module is not installed.

>:CHANnel<ch>:COLor
Function Sets the channel waveform color or queries the current setting.
Syntax :CHANnel<ch>:COLor {<NRf>,<NRf>,<NRf>}
>:CHANnel<ch>:COLor?
<NRf>: Red value 0 to 255
<NRf>: Green value 0 to 255
<NRf>: Blue value 0 to 255
Example :CHANNEL<ch>:COLor 0,255,255
>:CHANNEL<ch>:COLor? -> 0,255,255
Description Sets the waveform color used on the Acquisition Software or on Xviewer.

>:CHANnel<ch>:FREQ:INPut:BWIDTH
Function Sets or queries the bandwidth limit when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:INPut:BWIDTH {FULL|<Frequency>}
>:CHANnel<ch>:FREQ:INPut:BWIDTH?
<Frequency> = 100 Hz, 1 kHz, 10 kHz, 100 kHz
Example :CHANNEL<ch>:FREQ:INPUT:BWIDTH 100 Hz
Description An error occurs if a Frequency Module is not installed.
4.4 CHANnel Group

:CHANnel<ch>:FREQ:INPUT:CELImination

Function
Sets or queries chattering elimination when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:CELImination{<Time>}
:CHANnel<ch>:FREQ:INPUT:CELImination?<Time>

Example
:CHANNEL<ch>:FREQ:INPUT:CELImination 100ms
:CHANNEL<ch>:FREQ:INPUT:CELImination? -> 0.100

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:COUPling

Function
Sets or queries input coupling when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:COUPling {AC|DC}
:CHANnel<ch>:FREQ:INPUT:COUPling?

Example
:CHANNEL<ch>:FREQ:INPUT:COUPling DC
:CHANNEL<ch>:FREQ:INPUT:COUPling? -> DC

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:HYSTeresis

Function
Sets or queries hysteresis when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:HYSTeresis {HIGH|LOW|MIDDle}
:CHANnel<ch>:FREQ:INPUT:HYSTeresis?

Example
:CHANNEL<ch>:FREQ:INPUT:HYSTeresis LOW
:CHANNEL<ch>:FREQ:INPUT:HYSTeresis? -> LOW

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:PRESet

Function
Sets or queries the preset when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:PRESet {AC100v|AC200v|EMPichup|LOG12v|LOG24v|LOG3v|LOG5v|PULLup|USER|ZERO}

Example
:CHANNEL<ch>:FREQ:INPUT:PRESet USER

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:PROBe

Function
Sets or queries the probe attenuation when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:PROBe {<NRf>}
:CHANnel<ch>:FREQ:INPUT:PROBe?<NRf>

Example
:CHANNEL<ch>:FREQ:INPUT:PROBE 10
:CHANNEL<ch>:FREQ:INPUT:PROBE? -> 10

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:PULLup

Function
Sets or queries the pull-up on/off state when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:PULLup {<boolean>}
:CHANnel<ch>:FREQ:INPUT:PULLup?

Example
:CHANNEL<ch>:FREQ:INPUT:PULLUP ON
:CHANNEL<ch>:FREQ:INPUT:PULLUP? -> 1

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:SLOPe

Function
Sets or queries the slope when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:SLOPe {FALL|RISE}
:CHANnel<ch>:FREQ:INPUT:SLOPe?

Example
:CHANNEL<ch>:FREQ:INPUT:SLOPE FALL
:CHANNEL<ch>:FREQ:INPUT:SLOPE? -> FALL

Description
An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:INPUT:THReshold

Function
Sets or queries the threshold level when a Frequency Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:FREQ:INPUT:THReshold {<Voltage>}
:CHANnel<ch>:FREQ:INPUT:THReshold?<Voltage>

Example
:CHANNEL<ch>:FREQ:INPUT:THRESHOLD 10
:CHANNEL<ch>:FREQ:INPUT:THRESHOLD? -> 10.000E+00

Description
An error occurs if a Frequency Module is not installed.
**:CHANnel<ch>:FREQ:INPUT:VRANGe**

**Function**
Sets or queries the voltage range when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:INPUT:VRANGe {<Voltage>}
:CHANnel<ch>:FREQ:INPUT:VRANGe?

**Example**
:CHANNEL<ch>:FREQ:INPUT:VRANGE 10
:CHANNEL<ch>:FREQ:INPUT:VRANGE? -> 10

**Description**
An error occurs if a Frequency Module is not installed.

**:CHANnel<ch>:FREQ:LSCale:AVALue**

**Function**
Sets or queries linear scaling coefficient A when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:LSCale:AVALue {<NRf>}
:CHANnel<ch>:FREQ:LSCale:AVALue?

**Example**
:CHANNEL<ch>:FREQ:LSCALE:AVALUE 10
:CHANNEL<ch>:FREQ:LSCALE:AVALUE? -> 10.0000E+00

**Description**
An error occurs if a Frequency Module is not installed.

**:CHANnel<ch>:FREQ:LSCale:BVALue**

**Function**
Sets or queries linear scaling coefficient B when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:LSCale:BVALue {<NRf>}
:CHANnel<ch>:FREQ:LSCale:BVALue?

**Example**
:CHANNEL<ch>:FREQ:LSCALE:BVALUE 10
:CHANNEL<ch>:FREQ:LSCALE:BVALUE? -> 10.0000E+00

**Description**
An error occurs if a Frequency Module is not installed.

**:CHANnel<ch>:FREQ:LSCale:GETMeasure**

**Function**
Measures the X values of P1 and P2 for linear scaling when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:LSCale:GETMeasure {P1X|P2X|P1Y|P2Y}

**Example**
:CHANNEL<ch>:FREQ:LSCALE:GETMEASURE P1X
:CHANNEL<ch>:FREQ:LSCALE:GETMEASURE P2Y

**Description**
An error occurs if a Frequency Module is not installed.

**:CHANnel<ch>:FREQ:LSCale:MODE**

**Function**
Sets or queries linear scaling when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:LSCale:MODE {AXB|OFF|P12}
:CHANnel<ch>:FREQ:LSCale:MODE?

**Example**
:CHANNEL<ch>:FREQ:LSCALE:MODE OFF

**Description**
An error occurs if a Frequency Module is not installed.

---

**:CHANnel<ch>:FREQ:OFFSet**

**Function**
Sets or queries the offset value when a Frequency Module is installed in the specified channel (slot).

**Syntax**
:CHANnel<ch>:FREQ:OFFSet {<NRf>|<Frequency>|<Time>}
:CHANnel<ch>:FREQ:OFFSet?

**Example**
:CHANNEL<ch>:FREQ:OFFSET 1
:CHANNEL<ch>:FREQ:OFFSET? -> 1.0000E+00

**Description**
An error occurs if a Frequency Module is not installed.
4.4 CHANnel Group

:CHANnel<ch>:FREQ:SETup:CFRequency
Function  Sets or queries the center frequency when a Frequency Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:FREQ:SETup:CFRequency {<Frequency>}
:CHANnel<ch>:FREQ:SETup:CFRequency? {<NRf>} = 50 Hz, 60 Hz, 400 Hz
Example  :CHANNEL<ch>:FREQ:SETUP:CFREQUENCY 50
Description  An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:DECeleration
Function  Sets or queries the on/off state of decelerating prediction when a Frequency Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:FREQ:SETup:DECeleration {<boolean>}
:CHANnel<ch>:FREQ:SETup:DECeleration?  
Example  :CHANNEL<ch>:FREQ:SETUP:DECELERATION ON
          :CHANNEL<ch>:FREQ:SETUP:DECELERATION?  -> 1
Description  An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:DPULse
Function  Sets or queries the distance per pulse when a Frequency Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:FREQ:SETup:DPULse {<NRf>}
:CHANnel<ch>:FREQ:SETup:DPULse? {<NRf>} = 9.9999E+30 to –9.9999E+30
Example  :CHANNEL<ch>:FREQ:SETUP:DPULSE 1e15
          :CHANNEL<ch>:FREQ:SETUP:DPULSE?  -> 1.00000E+15
Description  An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:SMoothing
Function  Sets or queries the on/off state of smoothing when a Frequency Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:FREQ:SETup:FILTer:SMoothing {<boolean>}
:CHANnel<ch>:FREQ:SETup:FILTer:SMoothing?  
Example  :CHANNEL<ch>:FREQ:SETUP:FILTER:SMOOTHING MODE ON
Description  An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:FILTer:PAverage
Function  Sets or queries the pulse average count when a Frequency Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:FREQ:SETup:FILTer:PAverage {<NRf>}
:CHANnel<ch>:FREQ:SETup:FILTer:PAverage? {<NRf>} = 1 to 4096
Example  :CHANNEL<ch>:FREQ:SETUP:FILTER:PAVERAGE VALUE 10
Description  An error occurs if a Frequency Module is not installed.
4.4 CHANnel Group

:CHANnel<ch>:FREQ:SETup:FUNCTION
Function Sets or queries the measuring mode when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: FUNCTION {FREQuency|RPM|RPS|PERiod|DUTY|FWIDth|PINteg|VELocity}
Example :CHANNEL<ch>:FREQ:SETup:FUNCTION?
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:LRESET
Function Sets or queries the over limit reset when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: LRESET {<boolean>}
Example :CHANNEL<ch>:FREQ:SETup: LRESET ON
-> 1
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:MPULSE
Function Sets or queries whether the measurement pulse is positive or negative when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: MPULSE {POSitive|NEGative}
Example :CHANNEL<ch>:FREQ:SETup: MPULSE POSITIVE
-> POSITIVE
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:PROTate
Function Sets or queries the number of pulses per rotation when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: PROTate {<NRf>}
Example :CHANNEL<ch>:FREQ:SETup: PROTate 10
-> 10
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:RESet
Function Resets the pulse count when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: RESet
Example :CHANNEL<ch>:FREQ:SETup: RESet
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:STOPpredict
Function Sets or queries the on/off state of stop prediction when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: STOPpredict {<NRf>|OFF}
Example :CHANNEL<ch>:FREQ:SETup: STOPpredict OFF
-> OFF
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:TUNIT
Function Sets or queries the time unit when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: TUNIT {HOUR|MIN|SEC}
Example :CHANNEL<ch>:FREQ:SETup: TUNIT SEC
-> SEC
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>:FREQ:SETup:UNIT
Function Sets or queries the pulse integration unit when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>:FREQ:SETup: UNIT {<String>}
Example :CHANNEL<ch>:FREQ:SETup: UNIT "AAA"
-> "AAA"
Description An error occurs if a Frequency Module is not installed.
4.4 CHANnel Group

:CHANnel<ch>: FREQ: SETUP: UPULse
Function Sets or queries the unit/pulse when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: FREQ: SETUP: UPULse {<NRf>}
:CHANnel<ch>: FREQ: SETUP: UPULse?
Example :CHANnel<ch>: FREQ: SETUP: UPULSE 1e15
:CHANnel<ch>: FREQ: SETUP: UPULSE?
-> 1.00000E+15
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>: FREQ: SETUP: VUNIT
Function Sets or queries the unit of velocity when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: FREQ: SETUP: VUNIT {<String>}
:CHANnel<ch>: FREQ: SETUP: VUNIT?
Example :CHANnel<ch>: FREQ: SETUP: VUNIT "BBB"
:CHANnel<ch>: FREQ: SETUP: VUNIT?
-> "BBB"
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>: FREQ: VDIV
Function Sets or queries the Value/Div when a Frequency Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: FREQ: VDIV {<NRf>|<Frequency>|<Time>}
:CHANnel<ch>: FREQ: VDIV?
Example :CHANnel<ch>: FREQ: VDIV 10
:CHANnel<ch>: FREQ: VDIV?
-> 10.0E+00
Description An error occurs if a Frequency Module is not installed.

:CHANnel<ch>: STRain: BALance: EXECute
Function Balances strain when a Strain Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: STRain: BALance: EXECute
Example :CHANnel<ch>: STRain: BALance: EXECute
Description Balances channels that are switched on with the :CHANnel<ch>: STRain: BALance: CHANnel<ch> command.
• An error occurs if a Strain Module is not installed.

:CHANnel<ch>: STRain: BWIDth
Function Sets or queries the filter when a Strain Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: STRain: BWIDth {FULL|<Frequency>}
:CHANnel<ch>: STRain: BWIDth?
Example :CHANnel<ch>: STRain: BWIDth FULL
:CHANnel<ch>: STRain: BWIDth?
-> FULL
Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>: STRain: EXCitation
Function Sets or queries the bridge voltage when a Strain Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: STRain: EXCitation {<Voltage>}
:CHANnel<ch>: STRain: EXCitation?
Example :CHANnel<ch>: STRain: EXCitation 2V
:CHANnel<ch>: STRain: EXCitation?
-> 2.000000E+00
Description An error occurs if a Strain Module is not installed.

:CHANnel<ch>: STRain: GFACTOR
Function Sets or queries the gauge factor when a Strain Module is installed in the specified channel (slot).
Syntax :CHANnel<ch>: STRain: GFACTOR {<NRf>}
:CHANnel<ch>: STRain: GFACTOR?
Example :CHANnel<ch>: STRain: GFACTOR 2.00
:CHANnel<ch>: STRain: GFACTOR?
-> 2.00
Description An error occurs if a Strain Module is not installed.
:CHANnel<ch>:STRain:INVert

Function
Sets or queries whether or not the display is inverted when a Strain Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:STRain:INVert {<boolean>}
:CHANnel<ch>:STRain:INVert?

Example
:CHANNEL<ch>:STRAIN:INVERT ON
:CHANNEL<ch>:STRAIN:INVERT? -> 1

Description
An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:AVALue

Function
Sets or queries linear scaling coefficient A when a Strain Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:STRain:LSCale:AVALue {<NRf>}
:CHANnel<ch>:STRain:LSCale:AVALue?

Example
:CHANNEL<ch>:STRAIN:LSCALE:AVALUE 10
:CHANNEL<ch>:STRAIN:LSCALE:AVALUE? -> 10.0000E+00

Description
An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:BVALue

Function
Sets or queries offset value B when a Strain Module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>:STRain:LSCale:BVALue {<NRf>}
:CHANnel<ch>:STRain:LSCale:BVALue?

Example
:CHANNEL<ch>:STRAIN:LSCALE:BVALUE 5
:CHANNEL<ch>:STRAIN:LSCALE:BVALUE? -> 5.00000E+00

Description
An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRain:LSCale:DISPlaytype:MODE

Function
Sets or queries the linear scaling method when a Strain Module is installed in the specified channel (slot). (The method can only be set to SHUNT with a Strain Module with DSUB, Shunt-Cal.)

Syntax
:CHANnel<ch>:STRain:LSCale:MODE {AXB|OFF|P12|SHUNT}
:CHANnel<ch>:STRain:LSCale:MODE?

Example
:CHANNEL<ch>:STRAIN:LSCALE:MODE AXB

Description
An error occurs if a Strain Module is not installed.
4.4 CHANnel Group

:CHANnel<ch>:STRAin:LSCale:{P1X|P1Y|P2X|P2Y}

Function  Sets or queries the X or Y value of P1 or P2 for linear scaling when a Strain Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:STRAin:LSCale:{P1X|P1Y|P2X|P2Y} {<NRf>}
:CHANnel<ch>:STRAin:LSCale:{P1X|P1Y|P2X|P2Y}?

For P1X and P2X, 
\(<\text{NRf}> = -9.9999E+30 \text{ to } 9.9999E+30\)
For P1Y and P2Y, 
\(<\text{NRf}> = -9.9999E+25 \text{ to } 9.9999E+25\)

Example  :CHANNEL<ch>:STRAIN:LSCALE:P1X 10  
:CHANNEL<ch>:STRAIN:LSCALE:P1X?
-> 10.0000E+00

Description  An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRAin:LSCale:SHUNT

Function  Executes shunt calibration when a Strain Module is installed in the specified channel (slot). (This command only works with a Strain Module with DSUB, Shunt-Cal.)
Syntax  :CHANnel<ch>:STRAin:LSCale:SHUNT
Example  :CHANNEL<ch>:STRAIN:LSCALE:SHUNT

Description  An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRAin:LSCale:UNIT

Function  Sets or queries the unit of measurement to attach to the result of linear scaling when a Strain Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:STRAin:LSCale:UNIT {<String>}
:CHANnel<ch>:STRAin:LSCale:UNIT?

Example  :CHANNEL<ch>:STRAIN:LSCALE:UNIT "X"  
:CHANNEL<ch>:STRAIN:LSCALE:UNIT? -> "X"

Description  An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRAin:RANGe

Function  Sets or queries the measuring range when a Strain Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:STRAin:RANGe {<NRf>}  
:CHANnel<ch>:STRAin:RANGe?

Example  :CHANNEL<ch>:STRAIN:RANGE 5000  
:CHANNEL<ch>:STRAIN:RANGE? -> 5000

Description  An error occurs if a Strain Module is not installed.

:CHANnel<ch>:STRAin:UNIT

Function  Sets or queries the unit of measurement when a Strain Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:STRAin:UNIT {MV|USTR}
Example  :CHANNEL<ch>:STRAIN:UNIT USTR

Description  An error occurs if a Strain Module is not installed.

:CHANnel<ch>:TEMPerature:BURNout

Function  Sets or queries whether or not burnout is detected when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:TEMPerature:BURNout {<boolean>}
Example  :CHANNEL<ch>:TEMPERATURE:BURNOUT ON  
:CHANNEL<ch>:TEMPERATURE:BURNOUT? -> 1

Description  An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:BWIDTH

Function  Sets or queries the bandwidth limit when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:TEMPerature:BWIDTH {FULL|<Frequency>}
Example  :CHANNEL<ch>:TEMPERATURE:BWIDTH 2.0HZ  
:CHANNEL<ch>:TEMPERATURE:BWIDTH? -> 2.0E+00

Description  An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

:CHANnel<ch>:TEMPerature:COUPling

Function  Sets or queries input coupling when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).
Syntax  :CHANnel<ch>:TEMPerature:COUPling {TC|DC|GND}
Example  :CHANNEL<ch>:TEMPERATURE:COUPlING DC  
:CHANNEL<ch>:TEMPERATURE:COUPling?

Description  An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.
### :CHANnel<ch>:TEMPerature:RJC

**Function**: Sets or queries the RJC when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>:TEMPerature:RJC {<boolean>}  
:CHANnel<ch>:TEMPerature:RJC?

**Example**:  
:CHANNEL<ch>:TEMPERATURE:RJC ON  
:CHANNEL<ch>:TEMPERATURE:RJC? -> 1

**Description**: An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

### :CHANnel<ch>:TEMPerature:TYPE

**Function**: Sets or queries the thermocouple type when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>:TEMPerature:TYPE {K|E|J|T|L|U|R|S|B|W|Au7fe}  
:CHANnel<ch>:TEMPerature:TYPE?

**Example**:  
:CHANNEL<ch>:TEMPERATURE:TYPE K  
:CHANNEL<ch>:TEMPERATURE:TYPE? -> K

**Description**: An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

### :CHANnel<ch>:TEMPerature:UNIT

**Function**: Sets or queries the unit of measurement values when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>:TEMPerature:UNIT {C|F|K}  
:CHANnel<ch>:TEMPerature:UNIT?

**Example**:  
:CHANNEL<ch>:TEMPERATURE:UNIT C  
:CHANNEL<ch>:TEMPERATURE:UNIT? -> C

**Description**: An error occurs if a Temperature, High Precision Voltage Isolation Module is not installed.

### :CHANnel<ch>[:VOLTage]:BWIDth

**Function**: Sets or queries the bandwidth limit when a Temperature, High Precision Voltage Isolation Module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>[:VOLTage]:BWIDth {FULL|<Frequency>}  
:CHANnel<ch>[:VOLTage]:BWIDth?

**Example**:  
:CHANNEL<ch>:VOLTAGE:BWIDTH FULL  
:CHANNEL<ch>:VOLTAGE:BWIDTH? -> FULL

**Description**: An error occurs if a voltage module is not installed.

### :CHANnel<ch>[:VOLTage]:COUPling

**Function**: Sets or queries input coupling when a voltage module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>[:VOLTage]:COUPling {AC|DC|GND|ACRMS|DCRMS|TC}  
:CHANnel<ch>[:VOLTage]:COUPling?

**Example**:  
:CHANNEL<ch>:VOLTAGE:COUPlING DC  
:CHANNEL<ch>:VOLTAGE:COUPlING? -> DC

**Description**:  
- An error occurs if a voltage module is not installed.  
- The following modules can be set to TC:  
  701265, 701261, 701262  
- The following module can be set to DCRMS or ACRMS:  
  70160  
- The following module cannot be set to AC:  
  701265

### :CHANnel<ch>[:VOLTage]:INVert

**Function**: Sets or queries whether or not the display is inverted when a voltage module is installed in the specified channel (slot).

**Syntax**:  
:CHANnel<ch>[:VOLTage]:INVert {<boolean>}  
:CHANnel<ch>[:VOLTage]:INVert?

**Example**:  
:CHANNEL<ch>:VOLTAGE:INVERT ON  
:CHANNEL<ch>:VOLTAGE:INVERT? -> 1

**Description**: An error occurs if a voltage module is not installed.
4.4 CHANnel Group

:CHANnel<ch>[:VOLTage]:LSCale:BVALUE

Function
Sets or queries linear scaling offset value B when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   BVALUE {<NRf>}
   :CHANnel<ch>[:VOLTage]:LSCale:BVALUE?
   <NRf> = -9.9999E+30 to 9.9999E+30

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:BVALUE 10
:CHANNEL<ch>:VOLTAGE:LSCALE:BVALUE?
-> 10.0000E+00

Description
An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:

DISPLAYtype:MODE

Function
Sets or queries the display format for linear scaling.

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:MODE {EXPonent|FLOating}
   :CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:MODE?

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:MODE EXPONENT
   :CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:MODE? -> EXPONENT

:CHANnel<ch>[:VOLTage]:LSCale:

DISPLAYtype:DECimalnum

Function
Sets or queries the decimal place when the display format for linear scaling is set to Floating.

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:DECimalnum {<NRf>|AUTO}
   :CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:DECimalnum?
   <NRf> = 0 to 3

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:DECIMALNUM AUTO
   :CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:DECIMALNUM? -> AUTO

:CHANnel<ch>[:VOLTage]:LSCale:

DISPLAYtype:SUBunit

Function
Sets or queries the sub unit when the display format for linear scaling is set to Floating.

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:SUBunit (AUTO|NONE|PICO|NANO|MICro|MILI|KILo|MEGA|GIGA|TERA)
   :CHANnel<ch>[:VOLTage]:LSCale:
   DISPLAYtype:SUBunit?

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:SUBUNIT AUTO
   :CHANNEL<ch>:VOLTAGE:LSCALE:
   DISPLAYTYPE:SUBUNIT? -> AUTO

:CHANnel<ch>[:VOLTage]:LSCale:

GETMeasure

Function
Measures the X values of P1 and P2 for linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   GETMeasure {P1X|P2X}

Example
:CHANNEL<ch>:VOLTAGE:GETMeasure P1X

Description
An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:

MODE

Function
Sets or queries linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:
   MODE {AXB|OFF|P12}
   :CHANnel<ch>[:VOLTage]:LSCale:
   MODE?

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:
   MODE AXB
   :CHANNEL<ch>:VOLTAGE:LSCALE:
   MODE? -> AXB

Description
An error occurs if a voltage module is not installed.
:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y}

Function
Sets or queries the X or Y value of P1 or P2 for linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:{P1X|P1Y|P2X|P2Y} {<NRf>}

For P1X and P2X,

\[
<NRf> = -9.9999E+30 \text{ to } 9.9999E+30
\]

For P1Y and P2Y,

\[
<NRf> = -9.9999E+25 \text{ to } 9.9999E+25
\]

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:P1X 10
:CHANNEL<ch>:VOLTAGE:LSCALE:P1X?
-> 10.0000E+00

Description
An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:LSCale:UNIT

Function
Sets or queries the unit of measurement to attach to the result of linear scaling when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:LSCale:UNIT {<String>}

Example
:CHANNEL<ch>:VOLTAGE:LSCALE:UNIT "RPM"
:CHANNEL<ch>:VOLTAGE:LSCALE:UNIT?
-> "RPM"

Description
An error occurs if a voltage module is not installed.

:CHANnel<ch>[:VOLTage]:PROBe

Function
Sets or queries the probe type when a voltage module is installed in the specified channel (slot).

Syntax
:CHANnel<ch>[:VOLTage]:PROBe {<NRf>|C10|C100}

Example
:CHANNEL<ch>:VOLTAGE:PROBE 10
:CHANNEL<ch>:VOLTAGE:PROBE?
-> 10

Description
An error occurs if a voltage module is not installed.
- Modules 701265, 701261, 701262 cannot be set or queried.
- The initial value is 10 (1 for module 701267).
4.5 GONogo Group

The GONogo group deals with GO/NO-GO judgment. GONogo group commands are only valid when the measuring mode is Triggered mode.

You cannot use the GO/NO-GO judgment function during synchronous operation.

**:GONogo:ACONDition**

Function: Sets or queries the GO/NO-GO judgment action condition.

**Syntax:**

```
:GONogo:ACONDITION {ALWAYS|FAILURE|SUCCESS}
:GONogo:ACONDITION?
```

**Example:**

```
:GONOGO:ACONDITON FAILURE
:GONOGO:ACONDITON -> FAILURE
```

**:GONogo:ACTION:BUZZer**

Function: Sets or queries whether or not a beep is sounded when the condition is met.

**Syntax:**

```
:GONogo:ACTION:BUZZer {<boolean>}
:GONogo:ACTION:BUZZer?
```

**Example:**

```
:GONOGO:ACTION:BUZZER OFF
:GONOGO:ACTION:BUZZER? -> 0
```

**:GONogo:ACTION:MAIL:ADDRESS**

Function: Sets or queries the destination e-mail address when the condition is met.

**Syntax:**

```
:GONogo:ACTION:MAIL:ADDRESS {<String>}
:GONogo:ACTION:MAIL:ADDRESS "yoko@yokogawa.jp.com"
:GONogo:ACTION:MAIL:ADDRESS? -> "yoko@yokogawa.jp.com"
```

**Description:** This command can be used when the optional Ethernet interface is installed.

**:GONogo:ACTION:MAIL:COUNT**

Function: Sets or queries the e-mail transmission limit when the condition is met.

**Syntax:**

```
:GONogo:ACTION:MAIL:COUNT {<NRf>}
:GONogo:ACTION:MAIL:COUNT?
```

**Example:**

```
:GONOGO:ACTION:MAIL:COUNT 100
:GONOGO:ACTION:MAIL:COUNT? -> 100
```

**Description:** This command can be used when the optional Ethernet interface is installed.

**:GONogo:ACTION:MAIL:MODE**

Function: Sets or queries whether or not an e-mail is sent when the condition is met.

**Syntax:**

```
:GONogo:ACTION:MAIL:MODE {<boolean>}
:GONogo:ACTION:MAIL:MODE?
```

**Example:**

```
:GONOGO:ACTION:MAIL:MODE OFF
:GONOGO:ACTION:MAIL:MODE? -> 0
```

**Description:** This command can be used when the optional Ethernet interface is installed.

**:GONogo:ACTION:SAVE[:MODE]**

Function: Sets or queries whether or not waveform data is saved to the storage media when the condition is met.

**Syntax:**

```
:GONogo:ACTION:SAVE{[:MODE]} {<boolean>}
:GONogo:ACTION:SAVE{[:MODE]}?
```

**Example:**

```
:GONOGO:ACTION:SAVE:MODE OFF
:GONOGO:ACTION:SAVE:MODE? -> 0
```

**Description:** Set or query the media type by using the :FILE:DIRECTORY:DRIVE command.

**:GONogo:ACTION:SAVE:TYPE**

Function: Sets or queries the data type for saving waveform data to the storage media when the condition is met.

**Syntax:**

```
:GONogo:ACTION:SAVE:TYPE {ASCII|BINARY|FLOAT}
:GONogo:ACTION:SAVE:TYPE?
```

**Example:**

```
:GONOGO:ACTION:SAVE:TYPE ASCII
```

**:GONogo:AREA**

Function: Sets or queries the waveform area that is judged.

**Syntax:**

```
:GONogo:AREA {CURSOR|FULL}
:GONogo:AREA?
```

**Example:**

```
:GONOGO:AREA FULL
:GONOGO:AREA? -> FULL
```

**:GONogo:COUNt?**

Function: Queries the number of performed GO/NO-GO judgments.

**Syntax:**

```
:GONogo:COUNt?
```

**Example:**

```
:GONOGO:COUNT? -> 10
```

**:GONogo:LOGic**

Function: Sets or queries the GO/NO-GO logical condition.

**Syntax:**

```
:GONogo:LOGic {AND|OR}
:GONogo:LOGic?
```

**Example:**

```
:GONOGO:LOGIC AND
:GONOGO:LOGIC? -> AND
```

**:GONogo:MODE**

Function: Sets or queries the GO/NO-GO judgment mode.

**Syntax:**

```
:GONogo:MODE {OFF|PARAMETER}
:GONogo:MODE?
```

**Example:**

```
:GONOGO:MODE PARAMETER
:GONOGO:MODE? -> PARAMETER
```

**Description:** Set the GO/NO-GO judgment to OFF during synchronous operation.
4.5 GONogo Group

:GONogo:NGCount?
Function Queries the GO/NO-GO judgment NO-GO count.
Syntax :GONogo:NGCount?
Example :GONOGO:NGCOUNT? -> 10

:GONogo:PARameter:ITEM<x>:CAUSe?
Function Queries whether or not the specified waveform parameter is the cause of a NO-GO judgment.
Syntax :GONogo:PARameter:ITEM<x>:CAUSe?
The <x> in ITEM<x> = 1 to 16
Example :GONogo:PARAMETER:ITEM1:CAUSE? -> 1
Description When the waveform parameter is the cause of a NO-GO judgment, the command returns 1. Otherwise, the command returns 0.

:GONogo:PARameter:ITEM<x>:MODE
Function Sets or queries whether or not the specified waveform parameter is OFF, or what its judgment criterion is.
Syntax :GONogo:PARameter:ITEM<x>:MODE {OFF|IN|OUT}
:GONogo:PARameter:ITEM<x>:MODE?
The <x> in ITEM<x> = 1 to 16
Example :GONogo:PARAMETER:ITEM1:MODE IN
:GONogo:PARAMETER:ITEM1:MODE? -> IN

:GONogo:PARameter:ITEM<x>:PARam?
Function Queries the waveform parameter of the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:PARam?
The <x> in ITEM<x> = 1 to 16
Example :GONogo:PARAMETER:ITEM1:PARAM? -> AVER

:GONogo:PARameter:ITEM<x>:TRACe
Function Sets or queries the channel number of the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:TRACe {<NRf>}
:GONogo:PARameter:ITEM<x>:TRACe?
The <x> in ITEM<x> = 1 to 16
The <x> in ITEM<x> = 1 to 1024
Example :GONogo:PARAMETER:ITEM1:TRACe 1
:GONogo:PARAMETER:ITEM1:TRACe? -> 1

:GONogo:PARameter:ITEM<x>:TYPE:
Function Sets or queries the upper and lower limits of the judgment area for the specified judgment condition.
Syntax :GONogo:PARameter:ITEM<x>:TYPE:
<Parameter> = [AMPLitude|AVERage|AVGFreq|AVGPeriod|BWIDth1|BWIDth2|DELay|DUTYcycle|FALL|FREQuency|HIGH|LOW|MAXimum|MIDDle|MINimum|NOVershoot|NWIDth|PERiod|PNUMber|POVershoot|PTOPeak|PWIDth|RISE|RMS|SDEViation|TY1Integ|TY2Integ|XY1Integ|XY2Integ]
Example :GONogo:PARAMETER:ITEM1:TYPE:AVERAGE 100MV,-100MV
:GONogo:PARAMETER:ITEM1:TYPE:AVERAGE? -> 100.000000E+00
Description Only the values of parameters that have been set with this command can be queried.

:GONogo:PARameter:ITEM<x>:VALue?
Function Queries the automated measurement value of the specified GO/NO-GO judgment parameter.
Syntax :GONogo:PARameter:ITEM<x>:VALue?
The <x> in ITEM<x> = 1 to 16
Example :GONogo:PARAMETER:ITEM1:VALue? -> 50.000000E+00
Description When the mode is set to OFF or when the value is otherwise unmeasurable, the command returns “NaN” (not a number).

:GONogo:RSTatus?
Function Queries the most recent GO/NO-GO judgment.
Syntax GONogo:RSTatus?
Example GONogo:RSTatus? -> 0
Description The command returns 0 when the judgment is GO and it returns 1 when the judgment is NO-GO.
4.6 MEASURE Group

The MEASURE group deals with the automated measurement of waveform parameters. MEASURE group commands are only valid when the measuring mode is Triggered mode.

:MEASURE:AREA
Function Sets or queries the automatically measured waveform area for the waveform parameters.
Syntax MEASURE:AREA {CURSor|FULL}
Description When FULL is specified, the entire history waveform memory is subject to computation. When CURSor is specified, only the range set with MEASURE:CRANGE is subject to computation.

:MEASURE:CHANnel<ch>:DPROximal:UNIT
Function Sets or queries the distal, mesial, and proximal points.
Syntax :MEASURE:CHANnel<ch>:DPROximal:UNIT {<Voltage>,<Voltage>,<Voltage>|<Current>,<Current>,<Current>|<NRf>,<NRf>,<NRf>}
Description The settable ranges of <Voltage>, <Current>, and <NRf> vary depending on the range and offset settings. For details, see the SL1000 High Speed Data Acquisition Unit User’s Manual.
Example :MEASURE:CHANnel1:DPROXIMAL:UNIT -50V,0V,50V
:MEASURE:CHANnel1:DPROXIMAL:UNIT? -> -50.000E+00,0.0E+00,50.000E+00

:MEASURE:CHANnel<ch>:DPROximal:MODE
Function Sets or queries the distal, mesial, and proximal point mode setting.
Syntax :MEASURE:CHANnel<ch>:DPROximal:MODE {PERCent|UNIT}
Example :MEASURE:CHANnel1:DPROXIMAL:MODE PERCENT
:MEASURE:CHANnel1:DPROXIMAL:MODE? -> PERCENT

:MEASURE:CHANnel<ch>:DPROximal:PERCent
Function Sets or queries the distal, mesial, and proximal points as percentages.
Syntax :MEASURE:CHANnel<ch>:DPROximal:PERCent {<NRf>,<NRf>,<NRf>}
Example :MEASURE:CHANnel1:DPROXIMAL:PERCENT 40,60,80
:MEASURE:CHANnel1:DPROXIMAL:PERCENT? -> 40.0,60.0,80.0

:MEASURE:CHANnel<ch>:METHod
Function Sets or queries the high/low point setting method.
Syntax :MEASURE:CHANnel<ch>:METHod {AUTO|MAXMin}
Example :MEASURE:CHANnel1:METHOD AUTO
:MEASURE:CHANnel1:METHOD? -> AUTO

:MEASURE:CHANnel<ch>:<Parameter>:STATE
Function Sets or queries the on/off state of the measurement of the specified waveform parameter.
Syntax :MEASURE:CHANnel<ch>:<Parameter>:STATE {<boolean>}
Example :MEASURE:CHANnel1:AVERAGE:STATE ON
:MEASURE:CHANnel1:AVERAGE:STATE? -> 1
Function
Queries the automated measurement value of the specified waveform parameter.

Syntax
:MEASure:CHANnel<ch>:<Parameter>:VALue? {<NRf>}

<NRf> = 1 to 48000

<Parameter> = {AMPLitude|AVERage|AVGFreq|AVGPeriod|BWIDth1|BWIDth2|DUTYcycle|FALL|FREQuency|HIGH|LOW|MAXimum|MIDDle|MINimum|NOVershoot|NWIDth|PERiod|PNUMber|POVershoot|PTOPeak|PWIDTH|RISE|RMS|SDEViation|TY1Integ|TY2Integ|XY1Integ|XY2Integ}

Example

Description
If the value is immeasurable, the command returns “NaN” (not a number). The “<NRf>” at the end is used to specify what parameter value to query. It indicates the order of the parameter value since statistical processing began. If the specified parameter does not exist, the command returns “NaN” (not a number).

<NRf> can be omitted. If a value is not entered for <NRf>, the most recent waveform parameter value in the memory will be queried. When a value is entered for <NRf>, the order starts with the newest waveform in the memory and increases as the waveform parameter values get older.

Function
Sets or queries the waveform parameter measurement range.

Syntax
:MEASure:CRANge {<NRf>,<NRf>}
:MEASure:CRANge?

<NRf> = 0 to 134217728

Example

Function
Sets or queries the waveform parameter automated measuring mode.

Syntax
:MEASure:MODE {OFF|ON}
:MEASure:MODE?

Example
:MEASURE:MODE ON
:MEASURE:MODE? -> ON

Function
Waits for the execution of waveform parameter automated measurement with a set timeout.

Syntax
:MEASure:WAIT? {<NRf>}

<NRf> = 1 to 36000 (the timeout specified in 100 ms intervals)

Example
:MEASURE:WAIT? 100 -> 1

Description
• The command returns 0 if the automated measurement finishes within the specified timeout. If automated measurement does not finish, or if it was never taking place to begin with, the command returns 1.
• Even if you set a long timeout, the command will return 0 as soon as automated measurement finishes.
### 4.7 TRIGger Group

TRIGger group commands are only valid when the measuring mode is Triggered mode.

#### :TRIGger:COMBination:CHANnel<ch>: COMBination

**Function**
Sets or queries the trigger AND/OR state of all of the bits in the specified channel for the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:CHANnel<ch>: COMBination {AND|OR}
```

**Example**
```
:TRIGGER:COMBINATION:CHANNEL<ch>: COMBINATION AND
```

#### :TRIGger:COMBination:CHANnel<ch>: HYSTeresis<x>

**Function**
Sets or queries the trigger hysteresis of the specified channel in the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:CHANnel<ch>: HYSTeresis<x> {HIGH|LOW|MIDDLE}
```

**Example**
```
```

#### :TRIGger:COMBination:CHANnel<ch>: LEVEL<x>

**Function**
Sets or queries the trigger level of the specified channel in the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:CHANnel<ch>: LEVEL{x} {<Voltage>|<NRf>|<Current>}
```

**Example**
```
:TRIGGER:COMBINATION:CHANNEL<ch>: LEVEL1 0V
```

#### :TRIGger:COMBination:CHANnel<ch>: TYPE

**Function**
Sets or queries the trigger type of the specified channel in the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:CHANnel<ch>: TYPE {OFF|RISE|FALL|HIGH|LOW|BISlope|WLIn|WLOut|WINIn|WINOut}
```

**Example**
```
:TRIGGER:COMBINATION:CHANNEL<ch>: TYPE LOW
```

#### :TRIGger:COMBination:EXTERNAL:TYPE

**Function**
Sets or queries the external trigger type of the specified channel in the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:EXTERNAL: TYPE {OFF|RISE|FALL|HIGH|LOW}
```

**Example**
```
:TRIGGER:COMBINATION:EXTERNAL:TYPE RISE
```

#### :TRIGger:COMBination:MODE

**Function**
Sets or queries the combination mode of the combination trigger class.

**Syntax**
```
:TRIGger:COMBination:MODE {AND|OR}
```

**Example**
```
:TRIGGER:COMBINATION:EXTERNAL:MODE OR
```
## 4.7 TRIGGER Group

### :TRIGGER:DELay

**Function**  
Sets or queries the delay (the time between the trigger point and the trigger position).

**Syntax**  
:TRIGGER:DELay \(<\text{Time}>\)
:TRIGGER:DELay?

\(<\text{Time}>\) = 0 to 10 s

**Example**  
:TRIGGER:DELAY 2US
:TRIGGER:DELAY? \(\rightarrow 2.00000E-06\)

**Description**  
The resolution depends on the sample rate. (The resolution is 0.1 divided by the sample rate.) However, the highest resolution that can be set is 10 nanoseconds (if the sample rate is higher than 10 MS/s, the resolution will be 10 nanoseconds). The value is fixed at 0 when an external clock is used.

### :TRIGGER:HOLDoff:TIME

**Function**  
Sets or queries the trigger hold off time.

**Syntax**  
:TRIGGER:HOLDoff:TIME \(<\text{Time}>\)
:TRIGGER:HOLDoff:TIME?

\(<\text{Time}>\) = 0 to 10 s (with a resolution of 10 nanoseconds)

**Example**  
:TRIGGER:HOLDOFF:TIME 500NS
:TRIGGER:HOLDOFF:TIME? \(\rightarrow 500.000E-09\)

**Description**  
The trigger hold-off time cannot be set or queried when the trigger source is set to EXTERNAL, LINE, or TIME.

### :TRIGGER[:SIMPLE]:HYSTEResis

**Function**  
Sets or queries the hysteresis of the simple trigger.

**Syntax**  
:TRIGGER[:SIMPLE]:HYSTEResis \{HIGH|LOW|MIDDle\}

:TRIGGER[:SIMPLE]:HYSTEResis?

**Example**  
:TRIGGER[:SIMPLE]:HYSTERESIS MIDDLE
:TRIGGER[:SIMPLE]:HYSTERESIS? \(\rightarrow \text{MIDDLE}\)

**Description**  
The hysteresis cannot be set or queried when the trigger source is set to EXTERNAL, LINE, or TIME.

### :TRIGGER[:SIMPLE]:LEVEL

**Function**  
Sets or queries the simple trigger of the channel specified by :TRIGGER[:SIMPLE]:SOURce.

**Syntax**  
:TRIGGER[:SIMPLE]:LEVEL \{<Voltage>|<NRf>|<Current>\}

:TRIGGER[:SIMPLE]:LEVEL?

**Example**  
:TRIGGER:SIMPLE:LEVEL 0V
:TRIGGER:SIMPLE:LEVEL? \(\rightarrow 0.0E+00\)

**Description**  
The trigger level settable ranges and resolutions:

<table>
<thead>
<tr>
<th>Input</th>
<th>Settable Range</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>(\pm (V/div) \times 10)</td>
<td>1/100 of the V/div value</td>
</tr>
<tr>
<td>Temperature</td>
<td>The measurement range of each type of thermocouple.</td>
<td>0.1 (For C, K, and F)</td>
</tr>
<tr>
<td>Strain</td>
<td>(\pm \text{(Measurable range)})</td>
<td>1 (\mu\text{STR}) or 0.0005 m/V</td>
</tr>
<tr>
<td>Acceleration</td>
<td>(\times 0.1 \pm 10\text{00000 unit})</td>
<td>0.01 unit</td>
</tr>
<tr>
<td>Frequency (701280)</td>
<td>(\pm (V/div) \times 10)</td>
<td>Equivalent to 0.0005 div or 0.001 div. The smallest value is 0.001 Hz.</td>
</tr>
<tr>
<td>RPM (701280)</td>
<td>(\pm (V/div) \times 10)</td>
<td>Equivalent to 0.0005 div or 0.001 div.</td>
</tr>
</tbody>
</table>

Example:

1 V/div = 0.01 V resolution
200 mV/div = 0.002 V
10 Hz/div = 0.005 Hz
1000 rpm/div = 0.5 rpm
2000 rpm/div = 2 rpm
5000 rpm/div = 5 rpm
### 4.7 TRIGger Group

<table>
<thead>
<tr>
<th>Input</th>
<th>Settable Range Range</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPS (701280)</td>
<td>± (V/div) × 10</td>
<td>Equivalent to 0.0005 div or 0.001 div. Example: 1 rps/div = 0.5 mrps resolution 2 rps/div = 0.002 rps 5 rps/div = 0.005 rps</td>
</tr>
<tr>
<td>Interval (701280)</td>
<td>± (V/div) × 10</td>
<td>Equivalent to 0.0005 div or 0.001 div. Example: 1 ms/div = 0.5 µs 2 ms/div = 2 µs 5 ms/div = 5 µs</td>
</tr>
<tr>
<td>Duty (701280)</td>
<td>± (V/div) × 10</td>
<td>Equivalent to 0.001 div 1%/div = 0.001% 2%/div = 0.002% 5%/div = 0.005%</td>
</tr>
<tr>
<td>Power source frequency (701280)</td>
<td>± (V/div) × 10</td>
<td>Equivalent to 0.0005 div or 0.001 div. Example: 1 Hz/div = 0.001 Hz 0.5 Hz/div = 0.0005 Hz 0.2 Hz/div = 0.0002 Hz 0.1 Hz/div = 0.0001 Hz</td>
</tr>
<tr>
<td>Pulse width (701280)</td>
<td>± (V/div) × 10</td>
<td>The float setting</td>
</tr>
<tr>
<td>Integration (701280)</td>
<td>± (V/div) × 10</td>
<td>The float setting</td>
</tr>
<tr>
<td>Speed (701280)</td>
<td>± (V/div) × 10</td>
<td>The float setting</td>
</tr>
</tbody>
</table>

#### :TRIGger[:SIMple]:SLOPe

**Function** Sets or queries the simple trigger type of the channel specified by :TRIGger[:SIMple]:SOURce.

**Syntax**

:TRIGger[:SIMple]:SLOPe {OFF|RISE|FALL|BISlope}

:TRIGger[:SIMple]:SLOPe?

**Example**

:TRIGGER:SIMPLE:SLOPE RISE
:TRIGGER:SIMPLE:SLOPE? -> RISE

**Description**

The hysteresis cannot be set or queried when the trigger source is set to EXTernal, LINE, or TIME.

#### :TRIGger[:SIMple]:SOURce

**Function** Sets or queries the simple trigger source.

**Syntax**

:TRIGger[:SIMple]:SOURce {<NRf>|EXTernal|LINE|OFF|TIME}

:TRIGger[:SIMple]:SOURce?

**Example**

:TRIGGER:SIMPLE:SOURce 1
:TRIGGER:SIMPLE:SOURce? -> 1

**Description**

Of the :TRIGger:SIMple commands, only the :TRIGger:SIMple:SOURce command requires that "SIMple" not be omitted. When performing synchronous operation, you can set the trigger source to the one unit. Set the other unit trigger source to OFF.

#### :TRIGger:TIMer:DATE

**Function** Sets or queries the date of the time trigger.

**Syntax**

:TRIGger:TIMer:DATE <String>

:TRIGger:TIMer:DATE?

**Example**

:TRIGGER:TIMER:DATE "2007/12/04"
:TRIGGER:TIMER:DATE? -> "2007/12/04"

#### :TRIGger:TIMer:INTERVal

**Function** Sets or queries the trigger interval of the time trigger.

**Syntax**

:TRIGger:TIMer:INTERVal {MIN1|MIN2|MIN3|MIN4|MIN5|MIN6|MIN7|MIN8|MIN9|MIN10|MIN15|MIN20|MIN9|MIN10|MIN15|MIN20|MIN25|MIN30|MIN40|MIN45|MIN50|HOUR1|HOUR2|HOUR3|HOUR4|HOUR5|HOUR6|HOUR7|HOUR8|HOUR9|HOUR10|HOUR11|HOUR12|HOUR18|HOUR24}

:TRIGger:TIMer:INTERVal?

**Example**

:TRIGGER:TIMER:INTERVAL HOUR1
:TRIGGER:TIMER:INTERVAL? -> HOUR1

#### :TRIGger:TIMer:TIME

**Function** Sets or queries the time of the time trigger.

**Syntax**

:TRIGger:TIMer:TIME <String>

:TRIGger:TIMer:TIME?

**Example**

:TRIGGER:TIMER:TIME "12:34:56"
:TRIGGER:TIMER:TIME? -> "12:34:56"

#### :TRIGger:TYPE

**Function** Sets or queries the trigger type.

**Syntax**

:TRIGger:TYPE {COMBination|SIMple}

:TRIGger:TYPE?

**Example**

:TRIGGER:TYPE SIMPLE
:TRIGGER:TYPE? -> SIMPLE
SxAPI functions return two kinds of errors: unit errors and library errors.

<table>
<thead>
<tr>
<th>Error Codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 9999</td>
<td>Unit errors. These errors are returned from the SL1000. For details, see section 5.2.</td>
</tr>
<tr>
<td>10000 and greater</td>
<td>Library errors. These errors are returned from SxAPI. For details, see section 3.19 and the descriptions below.</td>
</tr>
</tbody>
</table>

### Library errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error, closed properly</td>
</tr>
<tr>
<td>10001</td>
<td>Timeout</td>
</tr>
<tr>
<td>10002</td>
<td>Cannot find the target unit</td>
</tr>
<tr>
<td>10003</td>
<td>Open failed</td>
</tr>
<tr>
<td>10004</td>
<td>Not opened</td>
</tr>
<tr>
<td>10005</td>
<td>Already opened</td>
</tr>
<tr>
<td>10006</td>
<td>Environment error</td>
</tr>
<tr>
<td>10007</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td>10008</td>
<td>Send error</td>
</tr>
<tr>
<td>10009</td>
<td>Receive error</td>
</tr>
<tr>
<td>10010</td>
<td>The received data is not in block format.</td>
</tr>
<tr>
<td>10011</td>
<td>System error</td>
</tr>
<tr>
<td>10012</td>
<td>ID violation</td>
</tr>
<tr>
<td>10013</td>
<td>Communication command error</td>
</tr>
<tr>
<td>10014</td>
<td>Insufficient buffer</td>
</tr>
<tr>
<td>10016</td>
<td>Cannot find the target unit group</td>
</tr>
<tr>
<td>10017</td>
<td>Invalid unit group</td>
</tr>
<tr>
<td>10018</td>
<td>Handle type violation</td>
</tr>
<tr>
<td>10019</td>
<td>Handle error</td>
</tr>
<tr>
<td>10020</td>
<td>Cannot find handle</td>
</tr>
<tr>
<td>10021</td>
<td>Command string violation</td>
</tr>
<tr>
<td>10022</td>
<td>Data outside of the range has been specified.</td>
</tr>
<tr>
<td>10023</td>
<td>The specified data does not exist.</td>
</tr>
<tr>
<td>10024</td>
<td>Conflict error</td>
</tr>
<tr>
<td>10031</td>
<td>Internal error</td>
</tr>
</tbody>
</table>
5.2 Unit Errors

These errors are returned from the SL1000.

Communication Syntax Errors (100 to 199)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Syntax error. A syntax error other than one of the ones listed below.</td>
</tr>
<tr>
<td>103</td>
<td>Invalid separator. Separate parameters with a comma.</td>
</tr>
<tr>
<td>104</td>
<td>Data type error. Use the correct data type for each parameter.</td>
</tr>
<tr>
<td>105</td>
<td>GET not allowed. GET is not supported as a response to an interface message.</td>
</tr>
<tr>
<td>108</td>
<td>Parameter not allowed. Check the number of parameters.</td>
</tr>
<tr>
<td>109</td>
<td>Missing parameter. Be sure to include all necessary parameters.</td>
</tr>
<tr>
<td>111</td>
<td>Header separator error. Separate commands from parameters with a space.</td>
</tr>
<tr>
<td>112</td>
<td>Header separator error. Check the command length.</td>
</tr>
<tr>
<td>113</td>
<td>Undefined header. Confirm the command name.</td>
</tr>
<tr>
<td>114</td>
<td>Header suffix out of range. Check the command.</td>
</tr>
<tr>
<td>120</td>
<td>Numeric data error. A value must be specified where the syntax contains &lt;NRf&gt;.</td>
</tr>
<tr>
<td>123</td>
<td>Exponent too large. Where the syntax contains &lt;NR3&gt;, make the exponent that follows E smaller.</td>
</tr>
<tr>
<td>124</td>
<td>Too many digits. Limit numeric values to 255 digits or less.</td>
</tr>
<tr>
<td>128</td>
<td>Numeric data not allowed. Use a data type other than &lt;NRf&gt;.</td>
</tr>
<tr>
<td>131</td>
<td>Invalid suffix. Check the units where the syntax contains &lt;Voltage&gt;, &lt;Time&gt;, or &lt;Frequency&gt;.</td>
</tr>
<tr>
<td>134</td>
<td>Suffix too long. Check the units where the syntax contains &lt;Voltage&gt;, &lt;Time&gt;, or &lt;Frequency&gt;.</td>
</tr>
<tr>
<td>138</td>
<td>Suffix not allowed. Units of measurement can only be used where the syntax contains &lt;Voltage&gt;, &lt;Time&gt;, or &lt;Frequency&gt;.</td>
</tr>
<tr>
<td>141</td>
<td>Invalid character data. Be sure to select one of the listed choices when the syntax contains {...</td>
</tr>
<tr>
<td>144</td>
<td>Character data too long. Check the spelling of strings where the syntax contains {...</td>
</tr>
<tr>
<td>148</td>
<td>Character data not allowed. Use a data type other than {...</td>
</tr>
<tr>
<td>150</td>
<td>String data error. Enclose parameters with single or double quotation marks where the syntax contains &lt;String&gt;.</td>
</tr>
<tr>
<td>151</td>
<td>Invalid string data. The parameter is either too long, or it contains an unusable character.</td>
</tr>
<tr>
<td>158</td>
<td>String data not allowed. Use a data type other than &lt;String&gt;.</td>
</tr>
<tr>
<td>161</td>
<td>Invalid block data. &lt;Block data&gt; cannot be used.</td>
</tr>
<tr>
<td>168</td>
<td>Block data not allowed. &lt;Block data&gt; cannot be used.</td>
</tr>
<tr>
<td>171</td>
<td>Invalid expression. Mathematical operations cannot be used.</td>
</tr>
<tr>
<td>178</td>
<td>Expression data not allowed. Mathematical operations cannot be used.</td>
</tr>
<tr>
<td>181</td>
<td>Invalid outside macro definition. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
</tbody>
</table>
## 5.2 Unit Errors

### Communication Execution Errors (200 to 299)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>221</td>
<td>Setting conflict. Check settings that are related to each other.</td>
</tr>
<tr>
<td>222</td>
<td>Data out of range. Check the ranges of the settings.</td>
</tr>
<tr>
<td>223</td>
<td>Too much data. Check data byte lengths.</td>
</tr>
<tr>
<td>224</td>
<td>Illegal parameter value. Check the ranges of the settings.</td>
</tr>
<tr>
<td>241</td>
<td>Hardware missing. Check that the specified options are all installed.</td>
</tr>
<tr>
<td>260</td>
<td>Expression error. Mathematical operations cannot be used.</td>
</tr>
<tr>
<td>270</td>
<td>Macro error. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>272</td>
<td>Macro execution error. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>273</td>
<td>Illegal macro label. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>275</td>
<td>Macro definition too long. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>276</td>
<td>Macro recursion error. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>277</td>
<td>Macro redefinition not allowed. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
<tr>
<td>278</td>
<td>Macro header not found. Does not conform to the IEEE488.2 macro specifications.</td>
</tr>
</tbody>
</table>

### Communication Query Errors (400 to 499)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>410</td>
<td>Query INTERRUPTED. Check the transmission and reception order.</td>
</tr>
<tr>
<td>420</td>
<td>Query UNTERMINATED. Check the transmission and reception order.</td>
</tr>
<tr>
<td>430</td>
<td>Query DEADLOCKED. Transmission will be stopped.</td>
</tr>
<tr>
<td>440</td>
<td>Query UNTERMINATED after indefinite response.</td>
</tr>
</tbody>
</table>

### System Communication Error (912)

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>912</td>
<td>Fatal error in the communication driver. Contact customer service.</td>
</tr>
</tbody>
</table>
Appendix

Appendix 1 Sample Programs

Visual Basic 6.0

Initializing and Setting Measurement Conditions

Connecting and Opening

The following code connects through USB to the unit group whose ID = 0, and returns the
configurations of the units in the group to their factory default settings. It also specifies
the reception of a measurement end event.

' Connect through USB
Ret = SxInit(SxEvent1.hWnd, SX_WIRE_USB, "", hComm)

' Open the unit group whose group ID is 0.
Ret = SxOpenGroup(hComm, 0, hGrp)

' Return unit configurations to their factory default
' settings.
Ret = SxInitSetup(hGrp)

' Specify reception of a measurement end event.
Ret = SxCreateEvent(hGrp, 0, SX_EV_ACQ_STOP)

Setting Measurement Conditions

The following code sets the measurement conditions as follows:

Acquisition channels = CH1, CH2, and CH3
All probes = “1:1”
All channel measuring ranges = ± 10[V]
Measuring mode = “Triggered”
Trigger mode = “Single”
Sampling rate = 10 k [S/s]
Measuring time = 0.1 s
Trigger position = 10%
Trigger source = CH1
Trigger level = 2.5 V

' Turn CH1 to 3 measurement on and set the probe and range.
For i = 0 to 2
'  Measurement = On
Ret = SxSetAcqSwitch(SxChHndl(hGrp, i), 1)
'  Probe = "1:1"
Ret = SxSetControl(SxChHndl(hGrp, i), ":CHAN<ch>:PROB 1")
'  Range = 10 V
Ret = SxSetControl(SxChHndl(hGrp, i), ":CHAN<ch>:VDIV 1")
  (The value is set to 1/10 of the range because the range is
  specified using VDIV.)
Next i

' Measuring mode = "Triggered"
Ret = SxSetAcqMode(hGrp, SX_ACQ_TRIG)

' Trigger mode = "Single"
Ret = SxSetTrigMode(hGrp, SX_TRIG_SINGLE)

' Sample interval = 100 us (= 10 kS/s)
Ret = SxSetSamplingInterval(hGrp, 0.0001)

' Record length = 0.1 s
Ret = SxSetAcqSpan(hGrp, 0.1)
Appendix 1 Sample Programs

```
' Trigger position = 10%
  Ret = SxSetTrigPos(hGrp, 10.0)

' Trigger source = CH1
  Ret = SxSetControl(hGrp, ":TRIG:SIMP:SOUR CH1")

' Trigger level = 2.5 V
  Ret = SxSetControl(hGrp, ":TRIG:SIMP:LEV 2.5")
```

**Starting Measurement**
The code below sends a command to the units to start measuring.
```
' Start measurement
  Ret = SxAcqStart(hGrp)
```

**Saving Waveform Data**
The code below saves the acquired waveform data to a (WDF format) file.
```
' Save the most recent waveform data
  Ret = SxSaveAcqData(SxUnitHndl(hGrp, 0), -1)
  Caution: -1 indicates the most recent data.
```

**Closing**

**Closing and Disconnecting**
The code below deletes the event handle, closes the unit group, and disconnects.
```
' Delete the event handle
  Ret = SxDeleteEvent(hGrp)

' Close the unit group
  Ret = SxCloseGroup(hGrp)

' Disconnect
  Ret = SxExit(hComm)
```
**Visual Basic 2008**

**Initializing and Setting Measurement Conditions**

**Connecting and Opening**

The following code connects through USB to the unit group whose ID = 0, and returns the configurations of the units in the group to their factory default settings. It also specifies the reception of a measurement end event.

' Connect through USB
Ret = SxAPI1.Init(WIRE.USB, "", hComm)

' Open the unit group whose group ID is 0.
Ret = SxAPI1.OpenGroup(hComm, 0, hGrp)

' Return unit configurations to their factory default settings.
Ret = SxAPI1.InitSetup(hGrp)

' Specify reception of a measurement end event.
Ret = SxAPI1.CreateEvent(hGrp, EV.ACQ_STOP)

**Setting Measurement Conditions**

The following code sets the measurement conditions as follows:

- **Acquisition channels** = CH1, CH2, and CH3
- All probes = "1:1"
- All channel measuring ranges = ± 10[V]
- Measuring mode = "Triggered"
- Trigger mode = "Single"
- Sampling rate = 10 k [S/s]
- Measuring time = 0.1 s
- Trigger position = 10%
- Trigger source = CH1
- Trigger level = 2.5 V

'Turn CH1 to 3 measurement on and set the probe and range.
For i = 0 to 2
' Measurement = On
Ret = SxAPI1.SetAcqSwitch(SxAPI1.ChHndl(hGrp, i), 1)
' Probe = "1:1"
Ret = SxAPI1.SetControl(SxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:PROB 1")
' Range = 10 V
Ret = SxAPI1.SetControl(SxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:VDIV 1")
'The value is set to 1/10 of the range because the range is specified using VDIV.)
Next i

'Measuring mode = "Triggered"
Ret = SxAPI1.SetAcqMode(hGrp, ACQMODE.TRIG)

'Trigger mode = "Single"
Ret = SxAPI1.SetTrigMode(hGrp, TRIGMODE.SINGLE)

'Sample interval = 100 us (= 10 kS/s)
Ret = SxAPI1.SetSamplingInterval(hGrp, 0.0001)

'Record length = 0.1 s
Ret = SxAPI1.SetAcqSpan(hGrp, 0.1)

'Trigger position = 10%
Ret = SxAPI1.SetTrigPos(hGrp, 10.0)
Appendix 1  Sample Programs

' Trigger source = CH1
  Ret = SxAPI1.SetControl(hGrp, "*:TRIG:SIMP:SOUR CH1")

' Trigger level = 2.5 V
  Ret = SxAPI1.SetControl(hGrp, "*:TRIG:SIMP:LEV 2.5")

Starting Measurement
The code below sends a command to the units to start measuring.
' Start measurement
  Ret = SxAPI1.AcqStart(hGrp)

Saving Waveform Data
The code below saves the acquired waveform data to a (WDF format) file.
' Save the most recent waveform data
  Ret = SxAPI1.SaveAcqData(SxAPI1.UnitHndl(hGrp, 0), -1, "C:\Data\filename")
  Caution: -1 indicates the most recent data.

Closing
Closing and Disconnecting
The code below deletes the event handle, closes the unit group, and disconnects.
' Delete the event handle
  Ret = SxAPI1.DeleteEvent(hGrp)

' Close the unit group
  Ret = SxAPI1.CloseGroup(hGrp)

' Disconnect
  Ret = SxAPI1.Exit(hComm)
Visual C#

Initializing and Setting Measurement Conditions
Connecting and Opening

The following code connects through USB to the unit group whose ID = 0, and returns the configurations of the units in the group to their factory default settings. It also specifies the reception of a measurement end event.

' Connect through USB
Ret = sxAPI1.Init(WIRE.USB, "", ref hComm);

' Open the unit group whose group ID is 0.
Ret = sxAPI1.OpenGroup(hComm, 0, ref hGrp);

' Return unit configurations to their factory default settings.
Ret = sxAPI1.InitSetup(hGrp);

' Specify reception of a measurement end event.
Ret = sxAPI1.CreateEvent(hGrp, (uint)EV.ACQ_STOP);

Setting Measurement Conditions

The following code sets the measurement conditions as follows:

Acquisition channels = CH1, CH2, and CH3
All probes = "1:1"
All channel measuring ranges = ± 10[V]
Measurement mode = "Triggered"
Trigger mode = "Single"
Sampling rate = 10 k[S/s]
Measuring time = 0.1 s
Trigger position = 10%
Trigger source = CH1
Trigger level = 2.5 V

' Turn CH1 to 3 measurement on and set the probe and range.
for (i = 0; i < 3; i++)
{
    ' Measurement = On
    Ret = sxAPI1.SetAcqSwitch(sxAPI1.ChHndl(hGrp, i), BOOL.ON);
    ' Probe = "1:1"
    Ret = sxAPI1.SetControl(sxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:PROB 1")
    ' Range = 10 V
    Ret = sxAPI1.SetControl(sxAPI1.ChHndl(hGrp, i), ":CHAN<ch>:VDIV 1")
    ' (The value is set to 1/10 of the range because the range is specified using VDIV.)
}

' Measuring mode = "Triggered"
Ret = sxAPI1.SetAcqMode(hGrp, ACQMODE.TRIG);

' Trigger mode = "Single"
Ret = sxAPI1.SetTrigMode(hGrp, TRIGMODE.SINGLE);

' Sample interval = 100 us (= 10 kS/s)
Ret = sxAPI1.SetSamplingInterval(hGrp, 0.0001);

' Record length = 0.1 s
Ret = sxAPI1.SetAcqSpan(hGrp, 0.1);

' Trigger position = 10%
Ret = sxAPI1.SetTrigPos(hGrp, 10.0);
Appendix 1 Sample Programs

' Trigger source = CH1
  Ret = sxAPI1.SetControl(hGrp, "::TRIG:SIMP:SOUR CH1");

' Trigger level = 2.5 V
  Ret = sxAPI1.SetControl(hGrp, "::TRIG:SIMP:LEV 2.5");

Starting Measurement
The code below sends a command to the units to start measuring.
' Start measurement
  Ret = sxAPI1.AcqStart(hGrp);

Saving Waveform Data
The code below saves the acquired waveform data to a (WDF format) file.
' Save the most recent waveform data
  Ret = sxAPI1.SaveAcqData(sxAPI1.UnitHndl(hGrp, 0), -1, "C:\\Data\\ filename");
  Caution: -1 indicates the most recent data.

Closing
Closing and Disconnecting
The code below deletes the event handle, closes the unit group, and disconnects.
' Delete the event handle
  Ret = sxAPI1.DeleteEvent(hGrp);

' Close the unit group
  Ret = sxAPI1.CloseGroup(hGrp);

' Disconnect
  Ret = sxAPI1.Exit(hComm);
# Index

<table>
<thead>
<tr>
<th>A</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquisition data information</td>
<td>3-47</td>
</tr>
<tr>
<td>acquisition number, latest</td>
<td>3-47</td>
</tr>
<tr>
<td>asynchronous messages</td>
<td>1-4</td>
</tr>
<tr>
<td>auto naming</td>
<td>3-41</td>
</tr>
<tr>
<td>auto recording condition settings</td>
<td>3-3</td>
</tr>
<tr>
<td>auto recording controls</td>
<td>3-3</td>
</tr>
<tr>
<td>auto recording destination</td>
<td>3-34</td>
</tr>
<tr>
<td>auto recording on/off</td>
<td>3-26</td>
</tr>
<tr>
<td>B</td>
<td>Page</td>
</tr>
<tr>
<td>binary data, send</td>
<td>3-20</td>
</tr>
<tr>
<td>C</td>
<td>Page</td>
</tr>
<tr>
<td>calibration</td>
<td>3-55</td>
</tr>
<tr>
<td>channel handle</td>
<td>1-3, 3-14</td>
</tr>
<tr>
<td>channel label</td>
<td>3-33</td>
</tr>
<tr>
<td>channel number</td>
<td>3-15</td>
</tr>
<tr>
<td>commands, send</td>
<td>3-18</td>
</tr>
<tr>
<td>commands, send and receive</td>
<td>3-20, 3-21</td>
</tr>
<tr>
<td>comments</td>
<td>3-43</td>
</tr>
<tr>
<td>communication command controls</td>
<td>3-2</td>
</tr>
<tr>
<td>communication commands</td>
<td>4-1</td>
</tr>
<tr>
<td>communication execution errors</td>
<td>5-3</td>
</tr>
<tr>
<td>communication handle</td>
<td>1-3, 3-12</td>
</tr>
<tr>
<td>communication handle, retrieving</td>
<td>3-1</td>
</tr>
<tr>
<td>communication query errors</td>
<td>5-3</td>
</tr>
<tr>
<td>communication syntax errors</td>
<td>5-2</td>
</tr>
<tr>
<td>CPU temperature</td>
<td>3-56</td>
</tr>
<tr>
<td>current directory</td>
<td>3-63</td>
</tr>
<tr>
<td>current drive</td>
<td>3-62</td>
</tr>
<tr>
<td>D</td>
<td>Page</td>
</tr>
<tr>
<td>debugging</td>
<td>3-5</td>
</tr>
<tr>
<td>development environment</td>
<td>1-1</td>
</tr>
<tr>
<td>device information retrieval</td>
<td>3-1</td>
</tr>
<tr>
<td>E</td>
<td>Page</td>
</tr>
<tr>
<td>error code</td>
<td>3-61</td>
</tr>
<tr>
<td>event control</td>
<td>3-2</td>
</tr>
<tr>
<td>event handler</td>
<td>3-23</td>
</tr>
<tr>
<td>events</td>
<td>1-4, 2-1</td>
</tr>
<tr>
<td>exit</td>
<td>3-6</td>
</tr>
<tr>
<td>F</td>
<td>Page</td>
</tr>
<tr>
<td>file, delete</td>
<td>3-66</td>
</tr>
<tr>
<td>file, divide</td>
<td>3-46</td>
</tr>
<tr>
<td>file information</td>
<td>3-65</td>
</tr>
<tr>
<td>file name</td>
<td>3-41</td>
</tr>
<tr>
<td>file order</td>
<td>3-42</td>
</tr>
<tr>
<td>file structure</td>
<td>1-1</td>
</tr>
<tr>
<td>H</td>
<td>Page</td>
</tr>
<tr>
<td>handles</td>
<td>1-3</td>
</tr>
<tr>
<td>handles, closing</td>
<td>3-1</td>
</tr>
<tr>
<td>handles, opening</td>
<td>3-1</td>
</tr>
<tr>
<td>I</td>
<td>Page</td>
</tr>
<tr>
<td>initialization</td>
<td>3-1, 3-6</td>
</tr>
<tr>
<td>instantaneous values</td>
<td>3-53</td>
</tr>
<tr>
<td>internal media operations</td>
<td>3-5</td>
</tr>
<tr>
<td>K</td>
<td>Page</td>
</tr>
<tr>
<td>key lock</td>
<td>3-61</td>
</tr>
<tr>
<td>L</td>
<td>Page</td>
</tr>
<tr>
<td>library errors</td>
<td>5-1</td>
</tr>
<tr>
<td>M</td>
<td>Page</td>
</tr>
<tr>
<td>manual trigger</td>
<td>3-45</td>
</tr>
<tr>
<td>measured data, delete</td>
<td>3-4</td>
</tr>
<tr>
<td>measurement condition settings</td>
<td>3-2</td>
</tr>
<tr>
<td>measurement controls</td>
<td>3-3</td>
</tr>
<tr>
<td>measurement on/off</td>
<td>3-25</td>
</tr>
<tr>
<td>measurement, start</td>
<td>3-44</td>
</tr>
<tr>
<td>measurement, stop</td>
<td>3-44</td>
</tr>
<tr>
<td>measuring group handle</td>
<td>1-3, 3-13</td>
</tr>
<tr>
<td>measuring group number</td>
<td>3-16</td>
</tr>
<tr>
<td>measuring group settings</td>
<td>3-1</td>
</tr>
<tr>
<td>measuring mode</td>
<td>3-26</td>
</tr>
<tr>
<td>module handle</td>
<td>1-3, 3-14</td>
</tr>
<tr>
<td>module information</td>
<td>3-9</td>
</tr>
<tr>
<td>module number</td>
<td>3-15</td>
</tr>
<tr>
<td>O</td>
<td>Page</td>
</tr>
<tr>
<td>operating environment</td>
<td>1-1</td>
</tr>
<tr>
<td>P</td>
<td>Page</td>
</tr>
<tr>
<td>parameters, get received</td>
<td>3-21, 3-22</td>
</tr>
<tr>
<td>pre-trigger points</td>
<td>3-30</td>
</tr>
<tr>
<td>pre-trigger position</td>
<td>3-30</td>
</tr>
<tr>
<td>R</td>
<td>Page</td>
</tr>
<tr>
<td>record count</td>
<td>3-40</td>
</tr>
<tr>
<td>recording destination folder</td>
<td>3-40</td>
</tr>
<tr>
<td>recording interval</td>
<td>3-39</td>
</tr>
<tr>
<td>recording interval mode</td>
<td>3-38</td>
</tr>
<tr>
<td>recording interval points</td>
<td>3-39</td>
</tr>
<tr>
<td>recording, start</td>
<td>3-46</td>
</tr>
<tr>
<td>recording start condition</td>
<td>3-34</td>
</tr>
<tr>
<td>recording start time</td>
<td>3-35</td>
</tr>
<tr>
<td>recording stop</td>
<td>3-46</td>
</tr>
<tr>
<td>recording stop condition</td>
<td>3-36</td>
</tr>
<tr>
<td>recording stop time</td>
<td>3-37</td>
</tr>
<tr>
<td>recording time</td>
<td>3-57</td>
</tr>
<tr>
<td>record points</td>
<td>3-38</td>
</tr>
<tr>
<td>re-search</td>
<td>3-7</td>
</tr>
</tbody>
</table>
# Index

<table>
<thead>
<tr>
<th>S</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample points</td>
<td>3-28</td>
</tr>
<tr>
<td>sample rate</td>
<td>3-27</td>
</tr>
<tr>
<td>sampling clock</td>
<td>3-27</td>
</tr>
<tr>
<td>sampling time</td>
<td>3-29</td>
</tr>
<tr>
<td>self test, execute</td>
<td>3-56</td>
</tr>
<tr>
<td>setup data access</td>
<td>3-4</td>
</tr>
<tr>
<td>setup data, initialize</td>
<td>3-54</td>
</tr>
<tr>
<td>setup data, load</td>
<td>3-54</td>
</tr>
<tr>
<td>setup data, save</td>
<td>3-54</td>
</tr>
<tr>
<td>SNTP settings</td>
<td>3-60</td>
</tr>
<tr>
<td>syntax rules</td>
<td>4-1</td>
</tr>
<tr>
<td>system error</td>
<td>5-3</td>
</tr>
<tr>
<td>system-related functions</td>
<td>3-4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger count</td>
<td>3-32</td>
</tr>
<tr>
<td>trigger mode</td>
<td>3-29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>U</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit, close</td>
<td>3-11</td>
</tr>
<tr>
<td>unit errors</td>
<td>5-2</td>
</tr>
<tr>
<td>unit group, close</td>
<td>3-10</td>
</tr>
<tr>
<td>unit group handle</td>
<td>1-3, 3-12</td>
</tr>
<tr>
<td>unit group information</td>
<td>3-8</td>
</tr>
<tr>
<td>unit group number</td>
<td>3-15</td>
</tr>
<tr>
<td>unit group, open</td>
<td>3-10</td>
</tr>
<tr>
<td>unit handle</td>
<td>1-3, 3-12</td>
</tr>
<tr>
<td>unit information</td>
<td>3-9</td>
</tr>
<tr>
<td>unit number</td>
<td>3-15</td>
</tr>
<tr>
<td>unit, open</td>
<td>3-11</td>
</tr>
<tr>
<td>units, number of</td>
<td>3-8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>W</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>waveform data, delete</td>
<td>3-53</td>
</tr>
<tr>
<td>waveform data, get</td>
<td>3-48</td>
</tr>
<tr>
<td>WDF file</td>
<td>3-51</td>
</tr>
</tbody>
</table>