User's Manual

Model 707741
WE Control API for .Net

YOKOGAWA
Yokogawa Meters & Instruments Corporation

IM 707741-62E
2nd Edition

Foreword

Thank you for purchasing the WE Control API (Model 707741).
This user's manual describes only the usage and functions specific to the .Net development platform.
The contents in this manual are given with the premise that you will read the WE Control API User's Manual along with this manual.
For a description of the WE Control API, see the following manuals.

Manual Title	Manual No.
WE Control API User's Manual	IM 707741-61E (WeAPI.pdf)*

* Henceforth, all IM707741-61E means WeAPI.pdf.

To ensure correct use, please read this manual thoroughly before operation.

Notes

- **The contents of this manual describe the WE Control API Ver. 5.0.1.0. If you are using another version of the API, the information given in this manual may differ from that of API that you are using.**
- The contents of this manual are subject to change without prior notice as a result of continuing improvements to the instrument's performance and functions.
- 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 any or all of the contents of this manual without YOKOGAWA's permission is strictly prohibited.

Trademarks

- Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe and Acrobat are trademarks of Adobe Systems Incorporated.
- All other company and product names used in this manual are trademarks or registered trademarks of their respective companies.

Revisions

1st Edition: June 2003
2nd Edition: July 2004

2nd Edition: July 2004 (YK)

How to Read This Document

How to read the “Parameter”

The words (IN) and (OUT) that follow the parameter indicate whether the parameter is an input parameter or an output parameter.

How to read the “Example (Visual Basic)”

Variable “ret”

Indicates a variable that contains the returned value.

“value -> parameter value”

Indicates that the variable “value” is set to the parameter value.

Contents

1. Overview

This user's manual describes the interface functions for controlling the WE7000 on the .Net platform (.Net Compatible WE Control API).
The .Net Compatible WE Control API was created using the class library of the Microsoft Windows .Net environment. The API can be used on the development platforms of Microsoft Visual Basic .Net, Microsoft Visual C# .Net, and Microsoft Visual C++ .Net.
Below are the main features.

Class-Based Access
Classes corresponding to stations or modules are defined and used to control them (provides class libraries).

Use of the Plug & Play Mechanism
The plug & play mechanism implemented in the Control Software or existing API can be used for easy programming.

Opening of the Module Control Panel of the WE Control Software
The module control panels and trigger setting dialog box shown on the WE Control Software can be called from the API. These can be used easily to check or change the setup data.

Independent from the Communication Format
The WE7000 currently supports optical, Ethernet (10BASE-T and 100BASE-T), and serial (RS-232) communications. The API absorbs the differences in these communication formats so that application programs are independent of the communication format.

Support for Microsoft Windows XP Professional/Home Edition and Microsoft Windows 2000 Professional Service Pack 2 or Later
Supports Microsoft Windows XP Professional/Home Edition and Microsoft Windows 2000 Professional Service Pack 2 or later, which are operating systems that the Visual Studio .NET family supports.

List of Files Provided for .Net

File Name	Description
WeAPINet.dll	DLL file for the .Net development platform
WeAPINet.pdf (IM707741-62E)	WE Control API online manual for the .Net development platform (Adobe Acrobat Reader 3.0 or later required for opening the file.)
Sample Program	Sample programs for various modules (included in the Samples folder)
*.dll	DLL files for the WE Control API
WeEvent.ocx	Control used to process asynchronous messages (events)

By default, the files listed above excluding the DLL files are copied to the C:\Program Files\WE7000\API directory. DLL files are copied to the Windows\SYstem32 directory.

Note
The operation of the codes given in this manual are not guaranteed.

Supported OSs
Microsoft Windows XP Professional/Home Edition and Microsoft Windows 2000 Professional Service Pack 2 or later

Supported Development Platform
Microsoft Visual Studio .Net

2. Using the .Net Compatible WE Control API

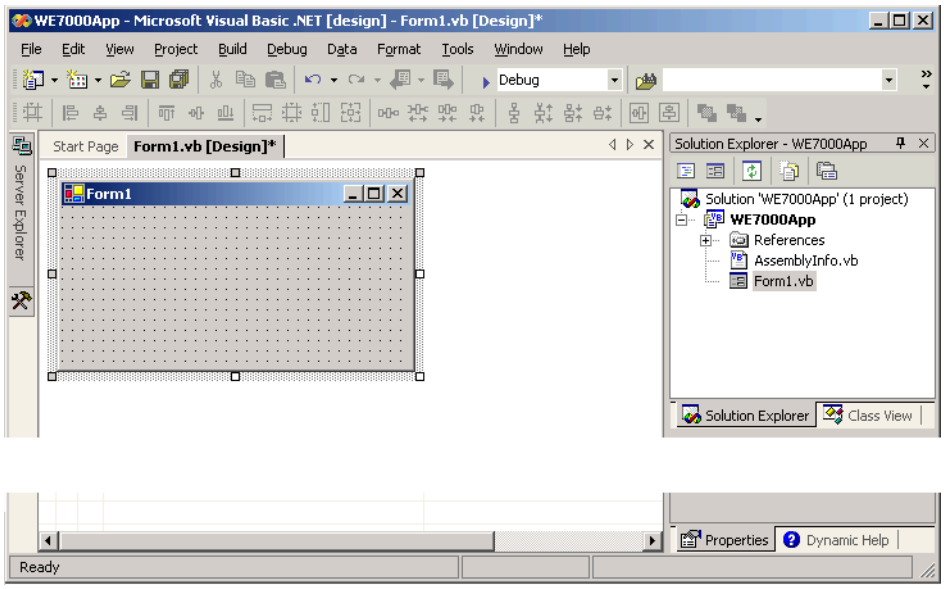
2.1 Program Model

The program model of the .Net Compatible WE Control API is the same as that of the existing WE Control API. For a description of the program model, see *chapter 3, "Programming Model" in the WE Control API User's Manual (IM707741-61E)*. Items that are specific to the .Net development platform are described below.

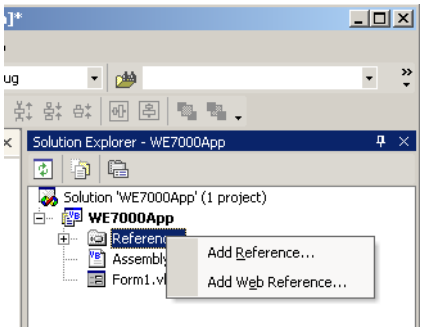
2.2 Class Library Reference

The .Net Compatible API is provided as a class library. On the .Net development platform, WeAPINet.Dll must be referenced. (Here, explanation is given for Visual Basic .Net.)
Below is the procedure for referencing the class library.

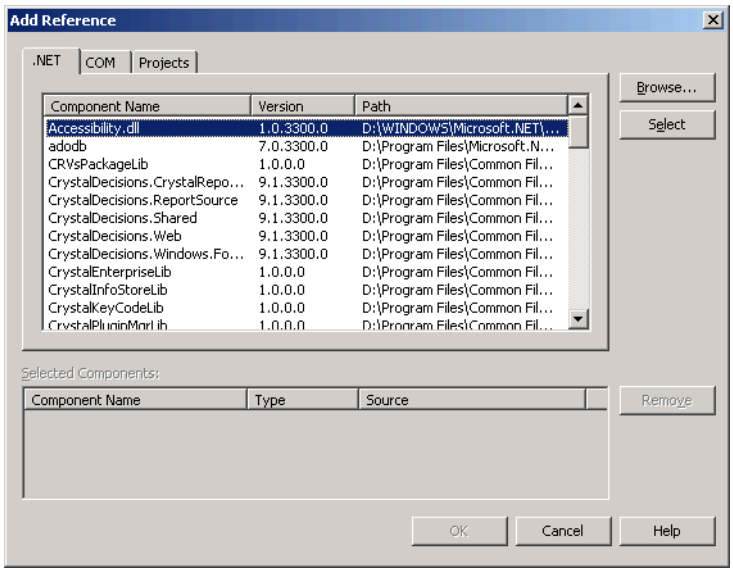
1. When you create a new solution, the window below opens.
   Right click **References** in the Solution Explorer displayed on the right.



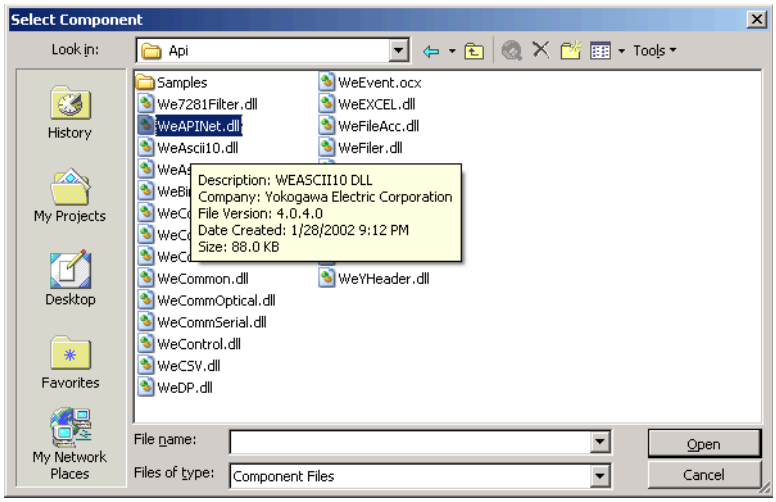
2. A shortcut menu appears. Click **Add Reference**.



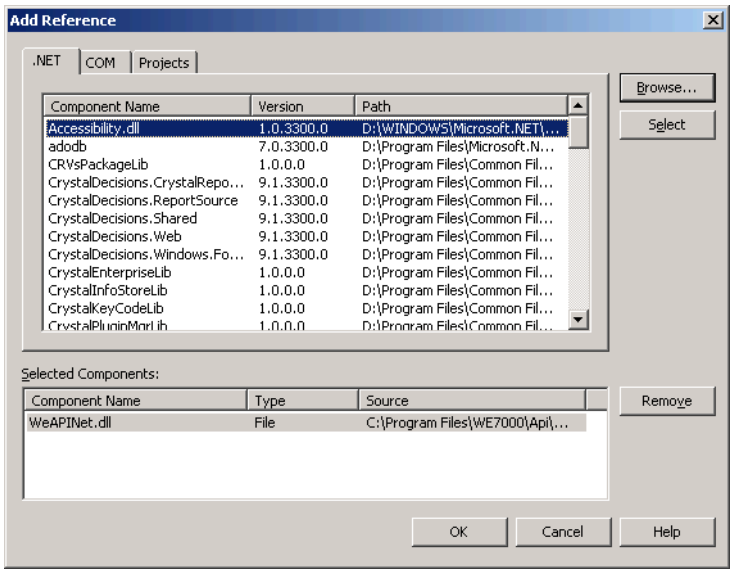
3. The Add Reference dialog box opens.
   Click **Browse**.



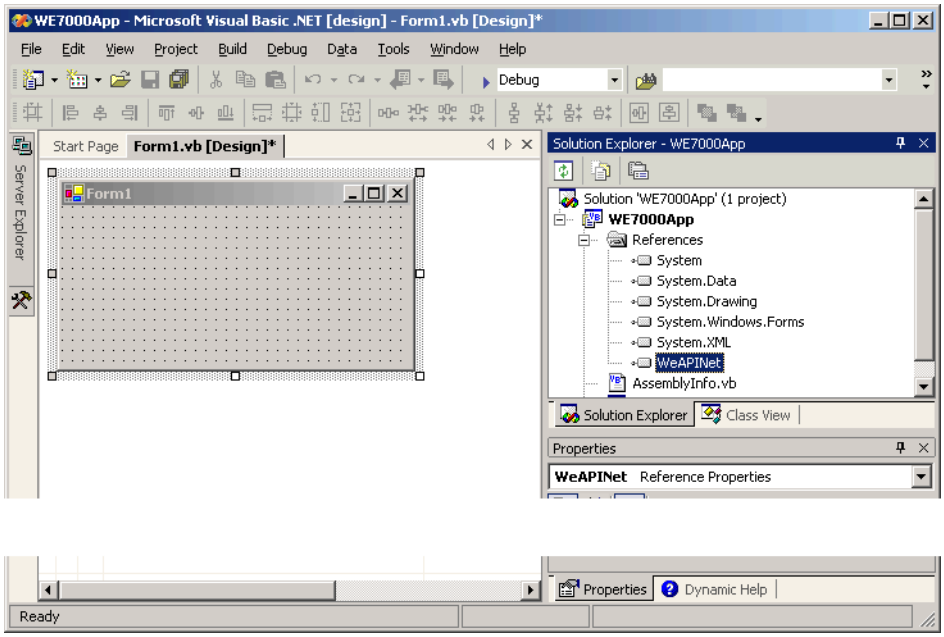
4. The Select Component dialog box opens. Select WeAPINet.dll and click **Open**. By default, WeAPINet.dll is located in C:\Program Files\WE7000\API.



5. Return to the Add Reference dialog box. WeAPINet.dll appears under Selected Components. Click **OK**.



6. WeAPINet is added to the Solution Explorer. And, the WeAPINet namespace is added to the solution.
   By declaring the required class in the code, the WE7000 can be controlled.



2.3 WEAPINet Classes

The WeAPINet.dll component includes the following classes.

Class Name	Explanation	Description
WeControl	Control class	Specifies the communication format. Various constants.
WeStation	Station class	Station control
WeModule	Module class	Module control
WeLib	Library class	Helper class
WeFilter	Filter class	Filters for the WE7281
WeFile	File class	Access wvf files.

The WeAPINet.dll component consists of the WEAPINet namespace.
This namespace includes the WeControl class, WeStation class, WeModule class, WeLib class, WeFilter class, WeFile class, and various structures.

2.4 WeAPINet Namespace Imports

In Visual Basic .Net, you can import namespaces using the Imports statement. Once a namespace is imported, you no longer have to write out the namespace. (This corresponds to the using directive in Visual C# .Net.)
The Imports statement is normally written before the Form class definition.

Example

```
' Import WeAPINet
Imports WeAPINet
Public Class Form1
    Inherits System.Windows.Forms.Form
```

2.5 Class Declaration

To control the WE7000, the class is declared.
Though it depends on the application that you are creating, if the WeControl class, WeStation class, or WeModule class is accessed in the form code, it is probably best to declare the class as a Form class member.

```
' Import WeAPINet
Imports WeAPINet
Public Class Form1
    Inherits System.Windows.Forms.Form
    ' WE7000 class declaration
    Private Comm As WeControl
    Private Station1 As WeStation
    Private WE7271 As WeModule
    Private WE7121 As WeModule
    ' WE classes are created at the beginning of the code that the Windows Form
     Designer generates.
#Region "Code that the Windows Form Designer generated"
    Public Sub New()
        MyBase.New()

        Comm = New WeControl()
        Station1 = New WeStation()
        WE7271 = New WeModule()
        WE7121 = New WeModule()
```

If WeAPINet is not imported, WeControl in the above example is written as follows:
Private Comm As WEAPINet.WeControl

2.6 Initialization and Termination

```
' Initialization. Selects the communication format.
' When using the WE7036 optical module
ret = Comm.Init(WeEvent1.hWnd, "optical devicename=we7036")
' Opens the station named Station 1.
ret = Station1.OpenStation("Station1")
' Turn the remote power of the station ON.
ret = Station1.Power(1)
' Opens the WE7271 installed to the first slot.
ret = WE7271.OpenModule(Station1, "WE7271:1", 1)
```

```
' Closes the handle.
ret = Station1.CloseHandle()
' Termination.
ret = Comm.Exit()
```

3. Detailed Explanation of Classes

3.1 Classes

WeControl Class

Method	Description	Page
Init	Initialization.	3-4
Exit	Termination.	3-5
GetStationList	Get the list of station names.	3-5

WeStation Class

Method	Description	Page
OpenStation	Open the station.	3-7
LinkStation	Open the measuring station link.	3-7
CloseHandle	Close the station.	3-8
GetStationInfo	Get station information.	3-8
Power	Turn ON/OFF the standby power to the measuring station.	3-9
ReStart	Restart the measuring station.	3-10
SetStationName	Set the station name and the comment.	3-10
GetStationName	Get the station name and the comment.	3-11
IdentifyStation	Identify the measuring station.	3-11
GetPower	Get the standby power ON/OFF state of the measuring station.	3-12
SetStatusLED	Set the STATUS LED.	3-12
GetStatusLED	Get the STATUS LED.	3-13
SetDIOConfig	Set the DIO configuration.	3-13
GetDIOConfig	Get the DIO configuration.	3-14
SetDIO	Set the DIO.	3-15
GetDIO	Get the DIO.	3-15
InitSetup	Initialize the current settings.	3-16
InitPreset	Replace preset values with default values.	3-16
SaveSetup	Save the current setup data to a file or update the preset values with the current settings.	3-17
LoadSetup	Load the setup data and update the current settings or update the current settings with the preset values.	3-17
ExecManualTrig	Generate a manual trigger.	3-18
ExecManualArming	Generate an arming signal.	3-19
SetTrigBusLogic	Set the trigger bus logic.	3-19
GetTrigBusLogic	Get the trigger bus logic.	3-20
SetEXTIO	Set the input/output of the trigger/time base input/output pin of the EXT. I/O connector.	3-20
GETEXTIO	Get the input/output setting of the trigger/time base input/output pin of the EXT. I/O connector.	3-21
SetTRIG	Set the trigger signal input and polarity from the TRIG terminal.	3-21
GetTRIG	Get the trigger signal input and polarity from the TRIG terminal.	3-22
SetTRIGIN	Set the trigger signal input and polarity from the TRIG IN terminal.	3-23
GetTRIGIN	Get the input and polarity of the trigger signal entering the TRIG IN terminal.	3-23
SetClockBusSource	Set the time base.	3-24
GetClockBusSource	Get the time base settings.	3-25
SetRcvTrigPacket	Set the receive station for trigger packets.	3-25
SetSndTrigPacket	Set the source station for trigger packets.	3-26
SetRcvClockPacket	Set the receive station for time base packets.	3-26
SetSndClockPacket	Set the source station for time base packets.	3-27
FireTrigPacket	Issue a trigger packet.	3-28
FireClockPacket	Issue a time base packet.	3-28
OutputEXTIOEvent	Set the event output of the EXT. I/O connector.	3-28
SetArmingSource	Set the arming source.	3-29
GetArmingSource	Get the arming source setting.	3-29
ShowTrigWindow	Show the trigger setting dialog box.	3-30
CloswTrigWindow	Close the trigger setting dialog box.	3-30
IsTrigWindow	Get the show/hide status of the trigger setting dialog box.	3-31
Start	Start the measurement module operation (by stations).	3-31
Stop	Stop the measurement module operation (by stations).	3-32

WeModule Class

WeLib Class

Method	Description	Page
ExecMeasureParam	Execute waveform parameter computation.	3-90
ExecMeasureParamAcqData	Execute waveform parameter computation using raw data.	3-91
GetHandle	Get the handle from the second parameter of the event.	3-92
IsNan	Check whether the data is non-numeric.	3-93
GetAlarmInfo	Get alarm information.	3-93
RtlMoveMemory	Copy memory contents.	3-94
TransAcqData	Convert acquisition data.	3-95

WeFilter Class

Method	Description	Page
Wvf2S16GetSize	Get the byte size when the waveform data in the specified file is converted to s16 format.	3-98
Wvf2W32GetSize	Get the byte size when the waveform data in the specified file is converted to w32 format.	3-99
Wvf2W7281GetSize	Get the byte size when the waveform data in the specified file is converted to w7281 format.	3-99
Wvf2S16	Convert the specified file to s16 format.	3-100
Wvf2W32	Convert the specified file to w32 format.	3-101
Wvf2W7281	Convert the specified file to w7281 format.	3-102

WeFile Class

Method	Description	Page
HeaderReadS	Read the header file of the single file.	3-103
DataRead	Read the data file of the single file.	3-104
HeaderWriteS	Write the header file of the single file.	3-105
DataWrite	Write the data file of the single file.	3-106
HeaderCsReadS	Read the header file of the sequential file.	3-107
CsRead	Read the data file of the sequential file.	3-108
HeaderCsWriteS	Write the header file of the sequential file.	3-109
CsWrite	Write the data file of the sequential file.	3-111
HeaderItemRead	Read the data of the specified item.	3-112
HeaderItemWrite	Write the data to the specified item.	3-113
GetSampleChNum	Get the number of samples and number of channels.	3-114
GetBlockNum	Get the number of blocks.	3-115
InitializeAcqInfo	Store the required data in the data information structure.	3-115

Note

- In the code examples of the detailed explanation of methods, it is assumed that WeControl, WeStation, WeModule, WeLib, and WeFilter classes are declared as Comm, Station, Module1, Lib1, Filter, and File, respectively.
- In the code examples of the detailed explanation of methods, it is assumed that variable ret that stores the return value of each method is declared as Short.
- The constants that appear in the detailed explanation of the methods are defined in the WeControl1 class.
  Example
  In the case of WE_CONTROLLER
  Ret = Comm.Init(WeOCX1.hWnd,”Ethernet”,WEControl.WE_CONTROLLER)
  As shown above, the constants are written as WEControl.XXXX (where XXXX is the definition symbol).
- Interfaces that are indicated as old interface in the explanations remain for compatibility with older versions.

3.2 WeControl Class

Init

Description

This is the function that carries out the initialization procedures. This method must be called first when using the WE Control API in an application program. Some of the activities that take place when this function is called include: the initialization of the network, the automatic detection of connected stations, and the initialization of the API environment.

Syntax

Init (*hWnd* As Integer, *comm* As String) As Short
Init (*hWnd* As Integer, *comm* As String, *type* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

hWnd (IN) — Window handle for receiving the WE7000 defined events (described later) that are generated by the station or module. For Visual Basic, specify the window handle of the WeEvent control (WeEvent.ocx). When creating a program without using WeEvent.ocx, specify 0 (NULL).

comm (IN) — The type of communication interface between the PC and meauring station.
"optical devicename = we7034" when using Optical Interface card WE7033/WE7034
"optical devicename = we7036" when using Optical Interface card WE7035/WE7036
"serial [option]" when using serial port (only for WE400/WE800)
Enter [option] settings as shown below. Be sure to specify an IP address and subnet mask.IP address: IP = IP address for the PC (ex. 192.168.21.128)
Subnet Mask: NETMASK = subnet mask (ex. 255.255.255.0)
Port Number (optional): PORTNO = port number (default is 34191)
Group number (optional): GROUPNO = group number (default is 0)
Transmission Mode (optional): COMPATIBLE = ON / OFF (default is ON)
[option] example: IP=192.168.21.128 NETMASK=255.255.255.0 PORTNO=34191 GROUPNO=1 COMPATIBLE=OFF
"ethernet [option]"
The option can be omitted. If you wish to enter settings, follow the instructions above. If you only have one Ethernet interface, it is not necessary to specify an IP address or subnet mask. When using more than one Ethernet interface, specify which Ethernet interface you are using in the IP address. It's also necessary to enter the subnet mask.
"USB" when using USB interface (only for WE500/WE900)
For detailed information regarding this option, see "Start Option" in the WE7000 PC-Based Measurement Instruments and WE7051 Ethernet Module*/WE7052 Fast Ethernet Module* User's Manuals.
* Only for WE400/WE800.

type (IN) Execution mode of the application program.
There are two execution modes: the controller mode in which the network and station can be controlled, and the server mode in which certain functions can be executed (store or monitor data, for example). The current version only supports the controller mode.
WE_CONTROLLER ' Controller
WE_SERVER ' Server
(On I/F that has type omitted, WE_CONTROLLER is set.)

Note:

Specify the window that will receive the WE7000 defined events in the hWnd parameter.
Regardless of the version of Windows, if more than one Ethernet card is installed you must specify an IP address and subnet mask.

Example (Visual Basic .Net)

```
' Receive the events with the WeEvent control (WeEvent.ocx).
' Initialize by specifying optical interface (WE7035/WE7036) and
' controller mode.
ret = Comm.Init (WeEvent1.hWnd, "optical devicename = we7036")
or
ret = Comm.Init (WeEvent1.hWnd, "optical devicename = we7036",
WeControl.WE_CONTROLLER)
```

Exit

Description

This is the method that carries out the termination procedures. Make sure to execute this method at the end of the application program.

Syntax

Exit () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Carry out termination procedures.
ret = Comm.Exit ()
```

GetStationList

Description

Gets the list of station names on a network.

Syntax

GetStationList (ByRef *num* As Short, *list*() As StationList) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (OUT) The number of measuring stations (includes the controller). The maximum number of measuring stations is defined by MaxStationNum.

list (OUT) The pointer to the structure containing the station name list.
The first member of the list contains controller information. The structure containing the station name list is as follows:

```
Structure StationList                          ' Structure containing the station
                                               ' name list
   name As String * MaxStationName             ' Station name
   addr As Integer                             ' Station's logical address
Structure Type
```

The maximum number of stations is defined by MaxStationNum.

Example (Visual Basic .Net)

```
' Get the list of station names.
Dim list(WeControl.MaxStationNum) As StationList
Dim num As Short
Num = 0
ret = Station.GetStationList (num, list)
```

3.3 WeStation Class

OpenStation

Description

Gets the station handle for controlling the station, by specifying the station name.

Syntax

OpenStation (*name* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

name (IN)	Station name. To get the handle that can be used to control all the measuring stations on the network, specify “BROADCAST.”

Note:

By opening, this method saves within the class the station handle used to identify the station.
The station handle is stored to variable member hSt.
If BROADCAST is specified, some methods in the class cannot be used (see the explanation of each method). In addition, if a network contains multiple measuring stations with the same name, only the handle of the closest measuring station can be obtained. Therefore, make sure to assign different names to measuring stations on the same network.

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station1."
ret = Station.OpenStation ("Station1")
```

LinkStation

Description

Gets the station link handle. The station link handle can be used to simultaneously control multiple stations.

Syntax

LinkStation (*num* As Short, *StList*() As WeStation) As Short
LinkStation (*num* As Short, *hStList*() As Integer) As Short

Old interface

LinkStation (*num* As Short, ByRef *hStList* As Integer) As Short ()

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (IN)	The number of stations to link
StList (IN)	An array of station classes you wish to link. The number of entries must adhere to the number specified by num.
hStList (IN)	An array of the station handles of the stations you wish to link. The number of entries must adhere to the number specified by num.

Example (Visual Basic .Net)

```
' Get the link handle of two stations, then turn ON the power.
Dim LinkSt As WeStation
Dim Station1 As WeStation
Dim Station2 As WeStation
Dim StList(1) As WeStation
' Open the measuring station with the name "Station 1."
ret = Staton1.OpenStation ("Station1")
' Open the measuring station with the name "Station 2."
ret = Station2.OpenStation ("Station2")
StList (0) = Station1
StList (1) = Station2
ret = LinkSt.LinkStation (2, StList)
ret = LinkSt.Power (WeControl.WE_ON)
```

CloseHandle

Description

Closes the measuring station.

Syntax

CloseHandle () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Note:

The station handle and link handle are released internally. Releasing the station handle also releases the module handles and module link handles within the measuring station.

Example (Visual Basic .Net)

```
' Close the measuring station with the name "Station 1."
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the module at slot 2 with a link number of 1.
ret = Module1.OpenModule (Station, "2", 1)
ret = Station.CloseHandle ()
```

GetStationInfo

Description

Gets the station information.

Syntax

GetStationInfo (ByRef *info* As StationInfoEx) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

info (OUT) The pointer to the extended measuring station information structure to be retrieced

The extended measuring station information structure is as follows:

```
Structure ModuleInfo                        ' Module information structure
    product As String * 8                   ' Product name(A string
                                            ' expressed as WExxxx.)
                                            '  (Example: WE7111)
    chNum As ushort                         ' Number of channels per
                                            ' module
    connect As Byte                         ' Number of links.  This number
                                            ' is 1if the module is not linked.
    connecttype As Byte                     ' Link condition
                                            ' WE_PARENT_MODULE
                                            ' The parent module of the link
                                            ' (The left most module of the
                                            ' link)
                                            ' WE_CHILD_MODULE
                                            ' The child module of the link
                                            ' WE_SINGLE_MODULE
                                            ' Module that is not linked.
    version As uint                         ' Software version of  the
                                            ' module driver
End Structure

Structure StationInfo                       ' Station information structure
    addr As ushort                          ' Station's logical address
    state As Byte                           ' Power ON/OFF state of the
                                            ' remote station
                                            ' WE_OFF: Off
                                            ' WE_ON: On
    moduleNum As Byte                       ' Number of modules that are
                                            ' inserted
    name As String * MaxStationName         ' Station name
    comment As String * MaxStationComment   ' Comment
End Structure
```

Example (Visual Basic .Net)

```
' Get the station information of Station 1.
' Open the measuring station with the name "Station 1."
Dim Info As StationInfoEx
Dim mdInfo(WeControl.MaxModuleNum - 1) As ModuleInfo
Info.mdInfo = mdInfo
ret = Station.OpenStation ("Station1")
ret = Station.GetStationInfo (info)
```

Power

Description

Turns ON/OFF the measuring station's power.

Syntax

Power (*sw* As Byte) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
sw (IN)	Switch state WE_OFF turns OFF the power. WE_ON turns ON the power.

Example (Visual Basic .Net)
```
' Turn ON Station 1.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.Power (WeControl.WE_ON)
```

Restart
Description
Restarts the station. The operation is similar to turning ON the power. However, the communication module does not restart.

Syntax
Restart () As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
None

Example (Visual Basic .Net)
```
' Restarts Station 1.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.Restart()
```

SetStationName
Description
Sets the station name and the comments for the measuring station.

Syntax
SetStationName (name As String, comment As String) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful. Attempting to set a station name that already exists results in an error.

Parameters
name (IN)	A string containing the station name (up to 256 characters) Use " " if you are not specifying a name. The default station name is comprised of the string “Station” + the logical address number (for example: Station 1).
comment (IN)	A string containing a comment (up to 256 characters) Use " " if you are not specifying a comment.

Example (Visual Basic .Net)

```
' Set "Plant 1" for the station name and "Measure Sample" for the comment.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetStationName ("Plant 1", "Measure Sample")
```

Note:

Executing this command overwrites the flash ROM of the measuring station. There is a limitation on the number of times the flash ROM can be overwritten (approx. 100,000 times). Therefore, minimize the execution of this command.

GetStationName

Description

Gets the station name and the comment for the measuring station.

Syntax

GetStationName (ByRef *name* As String, ByRef *comment* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

name (OUT)	Station name (up to 256 characters)
comment (OUT)	A string containing a comment (up to 256 characters)

Example (Visual Basic .Net)

```
' Gets the station name and the comment.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim strName As String
Dim strComment As String
ret = Station.GetStationName(strName,strComment)
```

IdentifyStation

Description

For measuring station identification, the LED of the optical interface module of the specified measuring station blinks. This is valid only when the optical interface module is being used as the communication interface.

Syntax

IdentifyStation () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Identify Station 1.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.IdentifyStation ()
```

GetPower

Description

Gets the power ON/OFF state of the station.

Syntax

GetPower (ByRef *sw* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

sw (OUT)	Power ON/OFF state of the remote station	
	WE_OFF	' The measuring station's power is OFF.
	WE_ON	' The measuring station's power is ON.

Example (Visual Basic .Net)

```
' Get the power's ON/OFF state of Station 1.
' Open the measuring station with the name "Station 1."
ret = Station.WeOpenStation ("Station 1")
Dim sw As Byte
ret = Station.WeGetPower (sw)
```

SetStatusLED

Description

Turns ON/OFF the measuring station's STATUS LED.

Syntax

SetStatusLED (*LEDNo* As Byte,*status* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

LEDNo (IN)	LED number	
	WE_LEDA	' LED A
	WE_LEDB	' LED B
status (IN)	LED setting	
	WE_OFF	' OFF
	WE_RED	' Illuminate in red
	WE_GRN	' Illuminate in green
	WE_ORG	' Illuminate in orange

Example (Visual Basic)

```
' Illuminate LED A of Station 1 in green.
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =SetStatusLED (WE_LEDA,WE_GRN)
```

Note:

This function can be used only on the WE500 and WE900.

GetStatusLED

Description

Gets the illumination status of the measuring station's STATUS LED.

Syntax

GetStatusLEDA (*LEDNo* As Byte,ByRef *status* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

LEDNo (IN)	LED number	
	WE_LEDA	' LED A
	WE_LEDB	' LED B
status (OUT)	LED A status	
	WE_OFF	' OFF
	WE_RED	' Illuminated in red
	WE_GRN	' Illuminated in green
	WE_ORG	' Illuminated in orange

Example (Visual Basic)

```
' Get the status of LED A on Station 1.
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =GetStatusLED (WE_LEDA,status)
```

Note:

This function can be used only on the WE500 and WE900.

SetDIOConfig

Description

Sets the input/output direction, the change detection polarity, and the change detection mask of the DIO pin of the measuring station's EXT IO.

Syntax

SetDIOConfig (*Pin* As Integer,*Direct* As Byte,*Pol* As Byte,*Mask* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

Pin(IN)	Pin number (0 to 3)
Direct (IN)	Input/output direction setting
	WE_KEEP ' Keep current setting
	WE_IN ' Input
	WE_OUT ' Output
Pol (IN)	Change detection polarity setting
	WE_KEEP ' Keep current setting
	WE_RISE 'Rise
	WE_FALL 'Fall
Mask (IN)	Change detection mask setting
	WE_KEEP ' Keep current setting
	WE_MASK ' Not detect (if this is specified, WE_EV_DIO# events will not be activated.)
	WE_DETECT ' Detect (if this is specified, WE_EV_DIO# (where # is the pin number) events will be activated. To generate an event, the event must also be defined using Create Event.)

Example (Visual Basic)

```
' Set conditions for pin number 1 on Station 1.
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =Power (WE_ON)
ret =CreateEvent (WE_EV_DIO1,erHandle)
ret =SetDIOConfig (1,WE_IN,WE_RISE,WE_DETECT)
```

Note:

This function can be used only on the WE500 and WE900.

GetDIOConfig

Description

Gets the input/output direction, the change detection polarity, and the change detection mask of the DIO pin of the measuring station's EXT IO.

Syntax

GetDIOConfig (*Pin* As Integer,ByRef *Direct* As Byte,ByRef *Pol* As Byte,ByRef *Mask* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

Pin (IN)	Pin number (0 to 3)
Direct (OUT)	Input/Output direction
	WE_IN ' Input
	WE_OUT ' Output
Pol (OUT)	Change detection polarity
	WE_RISE 'Rise
	WE_FALL 'Fall
Mask (OUT)	Change detection mask
	WE_MASK ' Not detect
	WE_DETECT ' Detect

Example (Visual Basic)

```
' Get conditions for pin number 1 on Station 1.
Dim ret As Integer
Dim Direct As Byte
Dim Pol As Byte
Dim Mask As Byte
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =GetDIOConfig(1,Direct,Pol,Mask)
```

Note:

This function can be used only on the WE500 and WE900.

SetDIO

Description

Sets the output on the DIO pin of the measuring station's EXT IO.

Syntax

SetDIO (*Pin* As Integer,*Val* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

Pin (IN)	Pin number (0 to 3)
Val (IN)	DIO output setting
	WE_KEEP ' Keep current setting
	WE_DIO_OFF ' Low status
	WE_DIO_ON ' High status

Example (Visual Basic)

```
' Output DIO pin number 1 on Station 1.
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =SetDIO (1,WE_DIO_ON)
```

Note:

This function can be used only on the WE500 and WE900.

GetDIO

Description

Gets the status of the DIO pin of the measuring station's EXT IO.

Syntax

GetDIO (*Pin* As Integer,ByRef *Val* As Byte)As Integer

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

Pin (IN)	Pin number (0 to 3)
Val (OUT)	DIO input/output status
	WE_DIO_OFF ' Low status
	WE_DIO_ON ' High status

Example (Visual Basic)

```
' Get the input/output status of DIO pin number 1 on Station 1.
' Open the measuring station with the name "Station1."
ret =StationOpenStation ("Station1")
ret =GetDIO (1,Val)
```

Note:

This function can be used only on the WE500 and WE900.

InitSetup

Description

Resets the current settings of the measuring station to default values.
The setup data of all modules in the measuring station and the trigger settings (synchronization between modules) of the measuring station are applicable.

Syntax

InitSetup () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Reset the setup data of the measuring station to the default values.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.InitSetup ()
```

InitPreset

Description

Update the current preset values of the measuring station to default values.
The setup data of all modules in the measuring station and the trigger settings (synchronization between modules) of the measuring station are applicable.

Syntax

InitPreset () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
None

Example (Visual Basic .Net)
```
' Update the preset values of the measuring station to default values.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.InitPreset ()
```

Note:
Executing this command overwrites the flash ROM of the measuring station. There is a limitation on the number of times the flash ROM can be overwritten (approx. 100,000 times). Therefore, minimize the execution of this command.

SaveSetup
Description
Saves the current setup data of the measuring station or updates the preset values. When saving to a file, the file is saved with a file extension ".set."

Syntax
SaveSetup (*filename* As String) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
filename (IN) File name. Specify the file without the file extension ".set." Specifying " " for the file name updates the preset values.

Example (Visual Basic .Net)
```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Save the current setup data to the file, c:\param.set.
ret = Station.SaveSetup ("c:\param")
' Update the preset values with the current setup data.
ret = Station.SaveSetup ("")
```

Note:
Executing this command when only " " is specified for the file name overwrites the flash ROM of the measuring station. There is a limitation on the number of times the flash ROM can be overwritten (approx. 100,000 times). Therefore, minimize the execution of this command.

LoadSetup
Description
Updates the current setup data of the measuring station using the setup data file or preset values.

Syntax
LoadSetup (*filename* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. The module configuration of the measuring station and the module configuration in the setup data file (including the installation position of the module) are compared. If they match, the setup data are updated (successful). If not, an error is returned.

Parameters

filename (IN)	File name. Specify the file without the file extension ".set." Specifying " " for the file name updates the current settings using preset values.

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Update the current setup data with the file, c:\param.set.
ret = Station.LoadSetup ("c:\param")
' Update the current setup data with the preset values.
ret = Station.LoadSetup ("")
```

ExecManualTrig

Description

Manually generates a trigger signal on the trigger bus that is common to all modules in the station. The trigger bus is used as a trigger source and the modules are triggered off of it.

Syntax

ExecManualTrig (*busNo* As Byte, *pulse* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

busNo (IN)	Trigger bus selection	
	WE_TRG1	' BUSTRG 1
	WE_TRG2	' BUSTRG 2
pulse (IN)	Trigger pulse selection	
	WE_TRGUP	' UP
	WE_TRGDOWN	' DOWM
	WE_TRGONESHOT	' One-shot pulse

Example (Visual Basic .Net)

```
' Generates a trigger signal on trigger bus "BUSTRG 1" using one-
' shot pulse.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.ExecManualTrig (WeControl.WE_TRG1, WeControl.WE_MANTRGONESHOT)
```

ExecManualArming

Description

Manually generates an arming signal.

Syntax

ExecManualArming () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Manually generate an arming signal.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.ExecManualArming ()
```

SetTrigBusLogic

Description

Sets the trigger condition of the trigger bus (AND/OR operation). This function is effective when multiple modules are driving the trigger bus. You can specify the trigger to occur when all trigger conditions are satisfied (AND operation) or when one of the trigger conditions is satisfied (OR operation).

Syntax

SetTrigBusLogic (*item* As Byte, *logic* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (IN)	Trigger bus selection	
	WE_TRG1	' Set bus trigger “BUSTRG 1.”
	WE_TRG2	' Set bus trigger “BUSTRG 2.”
logic (IN)	Logic selection	
	WE_TRGAND	' Set the bus logic to AND.
	WE_TRGOR	' Se the bus logic to OR.

Example (Visual Basic .Net)

```
' Set the logic of bus trigger "BUSTRG 1" to AND.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetTrigBusLogic (WeControl.WE_TRG1, WeControl.WE_TRGAND)
```

GetTrigBusLogic

Description

Gets the trigger condition of the trigger bus.

Syntax

GetTrigBusLogic (*item* As Byte, ByRef *logic* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (IN)	Trigger bus selection	
	WE_TRG1	' Get trigger bus “BUSTRG 1. “
	WE_TRG2	' Get trigger bus “BUSTRG 2. “
logic (OUT)	Trigger bus logic	
	WE_TRGAND	' Bus logic is AND.
	WE_TRGOR	' Bus logic is OR.

Example (Visual Basic .Net)

```
' Get the bus logic of bus trigger "BUSTRG 1."
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim logic As Byte
ret = Station.GetTrigBusLogic (WeControl.WE_TRG1, logic)
```

SetEXTIO

Description

Sets the input/output setting of the trigger signal I/O pin and the time base signal I/O pin of the EXT I/O connector located on the front panel of the station. These signal pins are bi-directional and can (1) pass the external signals to the trigger and clock buses in the station, (2) output the trigger signal or the clock, and (3) provide input signals for other stations. In effect, multiple stations can be synchronized.

Syntax

SetEXTIO (*trig1* As Byte, *trig2* As Byte, *clock* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

trig1 (IN)	Input/output setting of the trigger signal I/O 1 pin of the EXT I/O terminal	
	WE_TRGIN	' Use as an input pin. Pass the external signal to the ' trigger bus “BUSTRG 1.”
	WE_TRGOUT	' Use as an output pin. ' Output the trigger bus signal “BUSTRG 1.”
trig2 (IN)	Input/output setting of the trigger signal I/O 2 pin of the EXT I/O terminal	
	WE_TRGIN	' Use as an input pin. Pass the external signal to the ' trigger bus “BUSTRG 2.”
	WE_TRGOUT	' Use as an output pin. ' Output the trigger bus signal “BUSTRG 2.”
clock (IN)	Input/output setting of the time base signal I/O pin of the EXT I/O terminal	
	WE_CMNCLKOUT	' Use as an output pin.
	WE_CMNCLKIN	' Use as an input pin.

Example (Visual Basic .Net)

```
' Set the trigger signal I/O 1 pin to output, trigger signal I/O 2
' pin to input, and the time base signal I/O pin to input a common clock.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetEXTIO (WeControl.WE_EXTIOTRGOUT, WeControl.WE_EXTIOTRGIN,
WeControl.WE_CMNCLKIN)
```


GetEXTIO

Description

Gets the input/output setting of the trigger I/O pin and the timebase I/O pin of the EXT I/O connector located on the front panel of the station.

Syntax

GetEXTIO (ByRef *trig1* As Byte, ByRef *trig2* As Byte,ByRef *clock* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

trig1 (OUT)	Input/output setting of the trigger1 signal I/O 1 pin of the EXT I/O terminal	
	WE_TRGIN	' Use as an input pin. Pass the external signal to the ' trigger bus “BUSTRG 1.”
	WE_TRGOUT	' Use as an output pin. ' Output the trigger bus signal “BUSTRG 1.”
trig2 (OUT)	Input/output setting of the frigger2 signal I/O 2 pin of the EXT I/O terminal	
	WE_TRGIN	' Use as an input pin. Pass the external signal to the ' trigger bus “BUSTRG 2.”
	WE_TRGOUT	' Use as an output pin. ' Output the trigger bus signal “BUSTRG 2.”
clock (OUT)	Input/output setting of the time base signal I/O pin of the EXT I/O terminal	
	WE_CMNCLKOUT	' Use as an output pin.
	WE_CMNCLKIN	' Use as an input pin.

Example (Visual Basic .Net)

```
' Gets the input/output setting of the trigger signal I/O 1,
' trigger signal I/O 2, and
' time base signal I/O pins of the EXT I/O terminal.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim trig1 As Byte, trig2 As Byte
Dim clock As Byte
ret = Station.GetEXTIO (trig1, trig2, clock)
```

SetTRIG

Description

Sets the destination and polarity of the trigger bus signal entering the TRIG terminal located on the front panel of the measuring station.

Syntax

SetTRIG (*item* As Byte, *polarity* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (IN) Select the trigger bus destination of the signal entering the TRIG terminal.

WE_TRGNONE	' Do not pass/output signal to the trigger bus.
WE_TRG1IN	' Pass the signal to trigger bus “BUSTRG 1.”
WE_TRG2IN	' Pass the signal to trigger bus “BUSTRG 2.”
WE_BOTHIN	' Pass the signal to trigger buses “BUSTRG 1, 2.”
WE_TRG1OUT	' Output the signal to trigger bus “BUSTRG 1.”
WE_TRG2OUT	' Output the signal to trigger bus “BUSTRG 2.”

polarity (IN) Select the polarity of the input signal at the TRIG terminal

WE_TRGPOS	' Input/output the signal as is.
WE_TRGNEG	' Input/output the signal after inverting it.

Example (Visual Basic .Net)

```
' Pass/output the trigger signal entering the TRIG terminal to trigger buses
' "BUSTRG 1" and "BUSTRG 2" and set the polarity to positive.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetTRIG (WeControl.WE_BOTH, WeControl.WE_TRGPOS)
```

Note:

WE_TRG1OUT, WE_TRG2OUT, and WE_CMNCLKOUT are ignored on the WE400 or WE800. (The function ends normally, but does not work.)

GetTRIG

Description

Gets the destination and polarity of the trigger input signal entering the TRIG terminal located on the front panel of the station.

Syntax

GetTRIG (ByRef *item* As Byte, ByRef *polarity* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.3

Parameters

item (OUT) Select the trigger bus destination of the signal entering the TRIG terminal.

WE_TRGNONE	' Do not pass/output signal to the trigger bus.
WE_TRG1	' Pass the signal to trigger bus “BUSTRG 1.”
WE_TRG2	' Pass the signal to trigger bus “BUSTRG 2.”
WE_BOTH	' Pass the signal to trigger buses “BUSTRG 1, 2.”
WE_TRG1OUT	' Output the signal to trigger bus “BUSTRG 1.”
WE_TRG2OUT	' Output the signal to trigger bus “BUSTRG 2.”

polarity (OUT) Select the polarity of the input signal at the TRIG terminal

WE_TRGPOS	' Input the signal as is.
WE_TRGNEG	' Input the signal after inverting it.

Example (Visual Basic .Net)

```
' Get the destination and polarity of the trigger bus signal
' entering the TRIG terminal.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim item As Byte
Dim pol As Byte
ret = Station.GetTRIG (item, pol)
```

Note:

WE_TRG1OUT, WE_TRG2OUT, and WE_CMNCLKOUT are ignored on the WE400 or WE800. (The function ends normally, but does not work.)

SetTRIGIN

Description

Sets the destination and polarity of the trigger bus signal entering the TRIG IN terminal located on the front panel of the measuring station.

Syntax

SetTRIGIN (*item* As Byte, *polarity* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (IN)	Select the trigger bus destination of the signal entering the TRIG IN terminal.	
	WE_TRGNONE	' Do not pass signal to the trigger bus.
	WE_TRG1	' Pass the signal to trigger bus “BUSTRG 1. “
	WE_TRG2	' Pass the signal to trigger bus “BUSTRG 2. “
	WE_BOTH	' Pass the signal to trigger buses “BUSTRG 1, 2. “
polarity (IN)	Select the polarity of the input signal at the TRIG IN terminal	
	WE_TRGPOS	' Input the signal as is.
	WE_TRGNEG	' Input the signal after inverting it.

Example (Visual Basic .Net)

```
' Pass the trigger signal entering the TRIG IN terminal to trigger buses
' "BUSTRG 1" and "BUSTRG 2" and set the polarity to positive.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetTRIGIN (WeControl.WE_BOTH, WeControl.WE_TRGPOS)
```

Note:

Use this function when you need to maintain compatibility with the APIs for the WE400 and WE800. Use the SetTRIG function if you are creating a new program.

GetTRIGIN

Description

Gets the destination and polarity of the trigger input signal entering the TRIG IN terminal located on the front panel of the station.

Syntax

GetTRIGIN (ByRef *item* As Byte, ByRef *polarity* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (OUT)	TRIG IN terminal's bus	
	WE_TRGNONE	' Do not pass signal to the trigger bus.
	WE_TRG1	' Pass the signal to trigger bus "BUSTRG 1."
	WE_TRG2	' Pass the signal to trigger bus "BUSTRG 2."
	WE_BOTH	' Pass the signal to trigger buses "BUSTRG 1, 2."
polarity (OUT)	Polarity of the polarity of the input signal at the TRIG IN terminal	
	WE_TRGPOS	' Input the signal as is.
	WE_TRGNEG	' Input the signal after inverting it.

Example (Visual Basic .Net)

```
' Get the destination and polarity of the trigger bus signal
' entering the TRIG IN terminal.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim item As Byte
Dim pol As Byte
ret = Station.GetTRIGIN (item, pol)
```

Note:

Use this function when you need to maintain compatibility with the APIs for the WE400 and WE800. Use the GetTRIG function if you are creating a new program.

SetClockBusSource

Description

Sets the time base source that is output to the common clock bus (CMNCLK). The clock can only have one source.

Syntax

SetClockBusSource (*source* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

source (IN)	Time base source selection	
	WE_CMNCLKSRC_NONE	' No source
	WE_CMNCLKSRC_TRIGIN	' TRG IN terminal of the front panel
	WE_CMNCLKSRC_EXT.IO	' EXT I/O terminal of the front panel
	WE_CMNCLKSRC_SLOT0/WE_CMNCLKSRC_SLOT1/	
	WE_CMNCLKSRC_SLOT2/WE_CMNCLKSRC_SLOT3/	
	WE_CMNCLKSRC_SLOT4/WE_CMNCLKSRC_SLOT5/	
	WE_CMNCLKSRC_SLOT6/WE_CMNCLKSRC_SLOT7/	
	WE_CMNCLKSRC_SLOT8	' Slot 0 to slot 8

Example (Visual Basic .Net)

```
' Set the time base output of the module in slot 1 to be the time
' base source.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetClockBusSource (WeControl.WE_CMNCLKSRC_SLOT1)
```

GetClockBusSource

Description

Gets the time base source that is currently being output to the common clock bus.

Syntax

GetClockBusSource (ByRef *source* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

source (OUT) Timebase source

WE_CMNCLKSRC_NONE	' No source
WE_CMNCLKSRC_TRIGIN	' TRG IN terminal of the front panel
WE_CMNCLKSRC_EXT.IO	' EXT I/O terminal of the front panel
WE_CMNCLKSRC_SLOT0/WE_CMNCLKSRC_SLOT1/	
WE_CMNCLKSRC_SLOT2/WE_CMNCLKSRC_SLOT3/	
WE_CMNCLKSRC_SLOT4/WE_CMNCLKSRC_SLOT5/	
WE_CMNCLKSRC_SLOT6/WE_CMNCLKSRC_SLOT7/	
WE_CMNCLKSRC_SLOT8	' Slot 0 to slot 8

Example (Visual Basic .Net)

```
' Get the time base source setting.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim item As Byte
ret = Station.GetClockBusSource (item)
```

SetRcvTrigPacket

Description

Sets the stations that will receive the trigger packets that are used for packet triggering. Up to eight stations can be specified. FireTrigPacket () can be used to generate a trigger packet and send it to the stations specified here. The trigger packet can also be generated from the specified trigger source station using SetSndTrigPacket ().

Syntax

SetRcvTrigPacket (*num* As Short, *name*() As PacketList) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (IN)	Number of stations to specify (up to 8).
name (IN)	Pointer to the station name structure

```
Structure PacketList                ' Station name structure
    name As String                  ' Station name
End Structure
```

Example (Visual Basic .Net)

```
' Set station 1 and station 2 to receive the packet triggers.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim packet(WeControl.PacketListNum) As PacketList
packet.list (0).name = "Station 1"
packet.list (1).name = "Station 2"
ret = Station.SetRcvTrigPacket(2, packet)
```

SetSndTrigPacket

Description

Sets the trigger packet source stations to be used for packet triggering. Up to eight stations can be specified. When trigger bus “BUSTRG 1” or “BUSTRG 2” becomes active in the source station, the station generates the trigger packet. The receiving station outputs a trigger signal according to the trigger source to its own trigger bus “BUSTRG 1” or “BUSTRG 2.”

Syntax

SetSndTrigPacket (*num* As Integer, *name*() As PacketList) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (IN)	Number of stations (up to 8).
name (IN)	Pointer to the station name structure

```
Structure PacketList                ' Station name structure
    name As String                  ' Station name
End Structure
```

Example (Visual Basic .Net)

```
' Set Station 1 and Station 2 to send packet triggers.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim packet(WeControl.PacketListNum) As PacketList
packet.list (0).name = "Station 1"
packet.list (1).name = "Station 2"
ret = Station.SetSndTrigPacket(2, packet)
```

SetRcvClockPacket

Description

Sets the time base packet receiving measuring for packet time base. Up to eight receiving measuring stations can be specified. The measuring station receiving the time base packet generates one pulse of time base signal on its own clock bus for each packet.

Syntax

SetRcvClockPacket (*num* As Integer, *name*() As PacketList) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (IN)	Number of stations (up to 8)
name (IN)	Pointer to the station name structure

```
Structure PacketList        ' Station name structure
    name As String          ' Station name
End Structure
```

Example (Visual Basic .Net)

```
' Set Station 1 and Station 2 to receive time base packets.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim packet(WeControl.PacketListNum) As PacketList
packet.list (0).name = "Station 1"
packet.list (1).name = "Station 2"
ret = Station.SetRcvClockPacket(2, packet)
```

SetSndClockPacket

Description

Set the time base packet source for packet time base. The specified measuring station generates time base packets using its own time base signal. Up to eight time base packet measuring stations can be specified.

Syntax

SetSndClockgPacket (*num* As Integer, *name*() As PacketList) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

num (IN)	Number of stations (up to 8)
name (IN)	Pointer to the station name structure

```
Structure PacketList            ' Station name structure
    name As String              ' Station name
End Structure PacketList
```

Example (Visual Basic .Net)

```
' Set Station 1 and Station 2 to send time base packets.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim packet(WeControl.PacketListNum) As PacketList
packet.list (0).name = "Station 1"
packet.list (1).name = "Station 2"
ret = Station.SetSndClockPacket(2, packet)
```

FireTrigPacket

Description

Generates a trigger packet using software for packet triggering. The trigger packets are sent to the measuring stations specified by SetRcvTrigPacket ().

Syntax

FireTrigPacket (*item* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (IN) Trigger bus selection. The trigger signal is output to the receiving station trigger bus specified by this parameter.

WE_TRG1	' Output to trigger bus “BUSTRG 1.”
WE_TRG2	' Output to trigger bus “BUSTRG 2.”

Example (Visual Basic .Net)

```
' Generate a trigger packet for Station 1.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.FireTrigPacket(WeControl.WE_TRG1)
```

FireClockPacket

Description

Generates time base packets using software for packet time base.

Syntax

FireClockPacket () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Generates a time base packet.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.FireClockPacket ()
```

OutputEXTIOEvent

Description

Outputs an event signal using software to the event output signal pin of the EXT I/O terminal located on the front panel of the measuring station.

Syntax

OutputEXTIOEvent (*pulse* As Byte) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
pulse (IN) Output selection

WE_EXTIO_OFF	' OFF
WE_EXTIO_ON	' ON
WE_EXTIO_PULSE	' Pulse

Example (Visual Basic .Net)
```
' Turn ON the event signal output.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.OutputEXTIOEvent(WeControl.WE_EXTIO_ON)
```

Note:
If this function is used on the WE500 or WE900, the output of DIO pin 0 of the EXT. I/O is turned ON, and STATUS LED A illuminates in orange.

SetArmingSource
Description
Sets the arming signal source.

Syntax
SetArmingSource (*item* As Byte) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
item (IN) Arming signal source selection

WE_TRGNONE	' Do not select arming source.
WE_TRG1	' Set bus trigger signal “BUSTRG 1” as the arming ' signal source.
WE_TRG2	' Set bus trigger signal “BUSTRG 2” as the arming ' signal source.

Example (Visual Basic .Net)
```
' Set the bus trigger signal "BUSTRG 1" to the arming signal
' source.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.SetArmingSource (WeControl.WE_TRG1)
```

GetArmingSource
Description
Get the arming signal source setting.

Syntax
GetArmingSource (ByRef *item* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

item (OUT) Arming source selection

WE_TRGNONE	' Do not select arming signal source.
WE_TRG1	' Set bus trigger signal “BUSTRG 1” as the arming ' signal source.
WE_TRG2	' Set bus trigger signal “BUSTRG 2” as the arming ' signal source.

Example (Visual Basic .Net)

```
' Gets the arming signal source setting.
Dim item As Byte
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.GetArmingSource (item)
```

ShowTrigWindow

Description

Displays the trigger source/timebase source/arming setting dialog box that is used to set the module synchronization function within the measuring station.

Syntax

ShowTrigWindow (ByRef hWnd As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

hWnd (OUT) Trigger source/time base source/arming setting dialog box handle. This can use to change the display position of the setting dialog box.

Example (Visual Basic .Net)

```
' Open the trigger source/time base source/arming setting dialog box.
Dim hWnd As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.ShowTrigWIndow (hWnd)
' Calls the WIN32API and moves the window position to x=520, y=220.
ret = SetWIndowPos (hWnd, 0, 520, 220, 0, 0, 1)
```

CloseTrigWindow

Description

Closes the trigger source/time base source/arming setting dialog box that is used to set the module synchronization function within the measuring station.

Syntax

CloseTrigWindow () As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
None

Example (Visual Basic .Net)
```
' Close the trigger source/time base source/arming setting dialog box.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Station.CloseTrigWindow ()
```

IsTrigWindow
Description
Queries whether or not the trigger source/time base source/arming setting dialog box, that is used to set the module synchronization method within the station, is open.

Syntax
IsTrigWindow (ByRef *sw* As Byte) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
sw (OUT)	Returns 1 if the trigger source/time base source/arming setting dialog box is open, 0 if not.

Example (Visual Basic .Net)
```
' Query whether or not the trigger source/time base source/arming
' setting dialog box is open.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
Dim sw As Byte
ret = Station.IsTrigWIndow (sw)
```

Start
Description
Starts the operation of the measurement modules in the station collectively.

Syntax
Start () As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Note:
This method only issues the start command to the module. For example, it does not synchronize to the termination of the acquisition on the acquisition module. If you need such termination process, use IsRun in the module class, which polls (monitors) the end of the execution.

Parameters
None

Example (Visual Basic .Net)
```
' Start WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Station.Start()
```

Stop
Description
Stops the operation of the measurement modules in the station collectively.

Syntax
Stop () As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
None

Example (Visual Basic .Net)
```
' Stop WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Station.Start ()
  ......
ret = Station.Stop ()
```

3.4 WeModule Class

OpenModule

Description

Opens the module for controlling the module.

Syntax

OpenModule (*Station* As WeStation, *name* As String, *connection* As Short) As Short
OpenModule (*hSt* As Integer, *name* As String, *connection* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

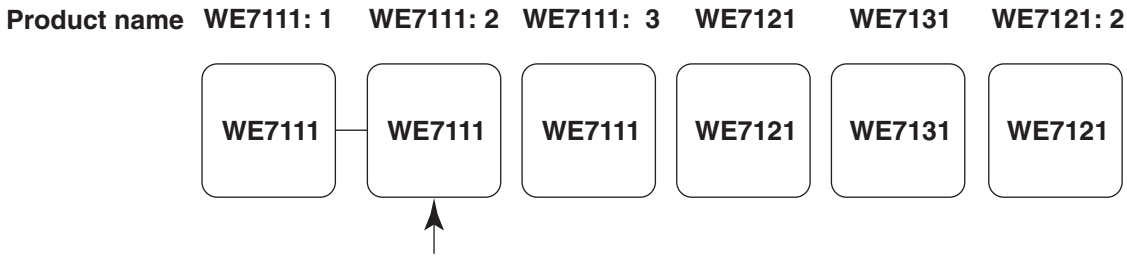
Parameters

Station (IN)	Station class
hSt (IN)	Station handle
name (IN)	Module's product name [:number] or slot number Using product name eliminates the need to know the slot number or position of the module in the station. The number inside the brackets indicates the position of the module counting the modules of the same type starting from the left most module of the same type (1 origin). (See the example below on how to specify the module's product name.) The brackets ( ) can be omitted in which case it is considered to be [:1]. To specify the 4-CH, 100 kS/s Isolated Digitizer Module WE7272, use "WE7271."
connection (IN)	The number of modules you wish to link. Specify 1 if you do not wish to link the modules.

Note:

By opening, this method saves within the class the module handle used to identify the module. The module handle is stored to variable member hMo.
By specifying the name parameter with the model's product name instead of the slot number, the program will be independent of the actual slot positions of the modules. Therefore, using the module's product name is encouraged. If the station is turned OFF, the module handles that have been obtained remains effective. This means that the same module handles can be used to control the modules when the measuring station is turned back ON. However, if the modules are switched or moved to different slot positions while the power is OFF, the behavior afterwards is not guaranteed.

Example of specifying the module's product name



Attempting to get the module handle of a child module that is linked to an opened module results in an error. Opening a child module that is linked to an unopened module clears the link.

Example (Visual Basic .Net)

- **Open the WE7111 module with a link number of 2 (when using class designation)**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Module1.OpenModule (Station, "WE7111:1", 2)
```

- **Open the WE7111 module with a link number of 2 (when using handle designation)**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Module1.OpenModule (Station.hSt, "WE7111:1", 2)
```

- **Open the module in slot 2 with a link number of 1**

```
  (When the "WE7111: 1" is not opened.)
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
ret = Module1.OpenModule (Station, "2", 1)
```

LinkModule

Description

Gets the module link handle. The module link handle can be used to the control multiple modules simultaneously.

Syntax

LinkModule (*num* As Short, *MoList*() As WeModule) As Short
LinkModule (*num* As Short, *hMoList*() As Integer) As Short
LinkModule (*num* As Short, ByRef *hMoList* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. Attempting to get a link handle for modules of a different type results in an error.

Parameters

num (IN)	The number of modules to link
MoList (IN)	An array of the module classes to be linked. The number of module handles in the array must equal the num parameter.
hMoList (OUT)	An array of the module classes of the modules to be linked. The number of module handles in the array must equal the num parameter.

Note:

By opening, this method saves within the class the module handle used to identify the module. The module handle is stored to variable member hMo.

Example (Visual Basic .Net)

```
Module1, Module2, and LinkModules are WeModule Class entities.
' Get the module link handle of two modules, then set the offset
' voltage to 1.234 V.
Dim MoList(1) As WeModule
' Get the station handle of the station named "Station 1"
ret = Station.OpenStation ("Station1")
' Open the module at slot 2 with a link number of 1.
ret = Module1.OpenModule (Station, "2", 1)
' Open the module at slot 4 with a link number of 1.
ret = Module2.OpenModule (Station, "4", 1)
MoList(0) = Module1
MoList(1) = Module2
ret = LinkModule.LinkModule (2, MoList)
ret = LinkModule.SetControl ("Offset", "1.234")
```

CloseHandle

Description

Closes the module.

Syntax

CloseHandle () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Note:

This method releases the module handle or the module link handle internally.

Example (Visual Basic .Net)

```
' Release the module handle of the module named "WE7111."
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the module at slot 2 with a link number of 1.
ret = Module1.OpenModule (Station, "WE7111:1", 1)
ret = Module1.CloseHandle ()
```

GetModuleInfo

Description

Gets the module information.

Syntax

GetModuleInfo (*no* As Short, ByRef *info* As ModuleInfoEx) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

no (IN) Link position of the module. The relative position counted starting from the parent module (0 origin). Specify 0 if the module is not linked.

info (OUT) Pointer to the module extended information structure that is returned

```
Structure ModuleInfoEx                          ' Module extended  Information
    mdInfo As ModuleInfo                        ' Module information
    maker As String* MaxMakerName               ' Name of the manufacturer
    productName As String* MaxProductName       ' Product name
End Structure
```

Note:

When the module is operating under a link, the module handle that was used to open the module is that of the parent module. Thus, the no parameter is necessary to specify the child module.

Example (Visual Basic .Net)

```
' Get the parent module information of the linked modules.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
Dim inf As ModuleInfoEx
ret = Module1.GetModuleInfo (0, inf)
```

InitSetup

Description

Resets the current module's setup data to the default values.

Syntax

InitSetup (*no* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

no (IN) Link position of the module. The relative position counted from the parent module (0 origin). Specify 0 if the module is not linked.

Example (Visual Basic .Net)

```
' Update the setup data of the parent module with the default values.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.InitSetup (0)
```

InitPreset

Description

Updates the preset values of the module with the default values.

Syntax

InitPreset (*no* As Short) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
no (IN) Link position of the module. The relative position counted starting from the parent module (0 origin). Specify 0 if the module is not linked.

Example (Visual Basic .Net)
```
' Update the preset values of the parent module with default values.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.InitPreset (0)
```

Note:
Executing this command overwrites the flash ROM of the measuring station. There is a limitation on the number of times the flash ROM can be overwritten (approx. 100,000 times). Therefore, minimize the execution of this command.

SaveSetup
Description
Saves the current setup data of the module or updates the preset values. When saving to a file, the file is saved with a file extension ".set."

Syntax
SaveSetup (*no* As Short, *filename* As String) As Short

Return value
Returns 0 if successful. Returns an error code if unsuccessful.

Parameters
no (IN) Link position of the module. The relative position counted starting from the parent module (0 origin). Specify 0 if the module is not linked.

filename (IN) File name. Specify the file without the file extension ".set." Specifying " " for the file name updates the preset values.

Example (Visual Basic .Net)
```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Save the current setup data to the file, c:\param.set.
ret = Module1.SaveSetup (0, "c:\param")
' Update the preset values with the current setup data.
ret = Module1.SaveSetup (0, "")
```

Note:
Executing this command when only " " is specified for the file name overwrites the flash ROM of the measuring station. There is a limitation on the number of times the flash ROM can be overwritten (approx. 100,000 times). Therefore, minimize the execution of this command.

LoadSetup

Description

Updates the current setup data of the station or module using the setup data file or preset values.

Syntax

LoadSetup (ByVal *no* As Short, ByVal *filename* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

no (IN)	Link position of the module. The relative position counted starting from the parent module (0 origin). Specify 0 if the module is not linked.
filename (IN)	File name. Specify the file without the file extension ".set." Specifying " " for the file name updates the current settings using preset values.

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Update the current setup data with the file, c:\param.set.
ret = Module1.LoadSetup (0, "c:\param")
' Update the current setup data with the preset values.
ret = Module1.LoadSetup (0, "")
```

CopySetup

Description

Copies the current setup data of the module to a specified module of the same type.

Syntax

CopySetup (*DesMo* As WeModule) As Short
CopySetup (*DesMo* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the module type of the copy source differs from that of the copy destination, an error is returned.

Parameters

DesMo (IN)	Module handle of the copy source
hDesMo (IN)	Module handle of the copy destination

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7111:1", 1)
' Open the second WE7111 module with a link number of 1.
ret = Module2.OpenModule (Station, "WE7111:2", 1)
' Copy the setup data of Module1 to Module2.
ret = Module1.CopySetup (Module2)
```

CopyChSetup

Description

Copies the current setup data of a channel of a module to the specified channel. This function is valid only when the modules are linked.

Syntax

CopyChSetup (*srcCh* As Short, *desCh* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

srcCh (IN)	Channel number of the copy source. One origin.
desCh (IN)	Channel number of the copy destination. One origin.

Note:

The slot number here signifies the relative slot number when the modules are linked.

Example (Visual Basic .Net)

```
' Copy the setup data of the WE7111 module at channel 1 (slot 1) to the
' module at channel 3 (slot 3).
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
ret = Module1.CopyChSetup (1, 3)
```

CopyChSetupEx

Description

Copies the current setup data of a channel of a module to the specified channel. This function can be used even when the modules are not linked.

Syntax

CopyChSetupEx (*srcCh* As Integer, *desStartCh* As Integer, *desEndCh* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

srcCh (IN)	Channel number of the copy source. One origin.
desStartCh (IN)	Channel number of the copy destination. One origin.
desEndCh (IN)	Last channel number of the copy destination. One origin.

Note:

The channel number here signifies the sequence number counted from the parent module when modules are linked.

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7251 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7251:1", 3)
' Copy the setup data of CH1 to the destination channels of CH2 through
' CH30.
ret = Module1.CopyChSetupEx (1, 2, 30)
```

SetControl

Description

Sets the individual control Parameters of a module.

Syntax

SetControl (*command* As String, *param* As Object) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN)	ASCII command name that is defined for each module. For details, see the command table of each measurement module.
param (IN)	Parameter dependent on the command

Note:

ASCII commands are defined for each module. For details on the ASCII commands of each module, see *chapter 8, "ASCII Commands" in the WE Control API User's Manual (IM 707741-61E)*.

Example (Visual Basic .Net)

```
' Set the offset value of the WE7111 module to 1.234 V.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
ret = Module1.SetControl ("Offset", "1.234")
```

GetControl

Description

Gets the current values of the individual control Parameters of the module.

Syntax

GetControl (*command* As String, ByRef *param* As Object) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN) ASCII command name that is defined for each module. For details, see the command table of each measurement module.

param (OUT) Parameter dependent on the command. In most cases, string data is returned. (There are exceptions.)

Note:

ASCII commands are defined for each module. For details on the ASCII commands of each module, see *chapter 8, "ASCII Commands" in the WE Control API User's Manual (IM 707741-61E)*.

Example (Visual Basic .Net)

```
' Get the offset value of the WE7111 module.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
' Variable used to retrieve data
Dim value As Object
ret = Module1.GetControl ("Offset", value)
```

SetControlEx

Description

Sets the individual control Parameters of the module. You can set the control parameters by specifying the data type.

Syntax

SetControlEx (*command* As String) As Short
SetControlEx (*command* As String, ByRef param As Byte) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As Byte) As Short
SetControlEx (*command* As String, ByRef param As SByte) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As SByte) As Short
SetControlEx (*command* As String, ByRef param As UInt16) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As UInt16) As Short
SetControlEx (*command* As String, ByRef param As Short) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As Short) As Short
SetControlEx (*command* As String, ByRef param As UInt32) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As UInt32) As Short
SetControlEx (*command* As String, ByRef param As Integer) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As Integer) As Short
SetControlEx (*command* As String, ByRef param As Single) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As Single) As Short
SetControlEx (*command* As String, ByRef param As Double) As Short
SetControlEx (*command* As String, *size* As Integer, ByRef *param*() As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN) ASCII command name that is defined for each module

size (IN) Number of data

param (IN) Data pointer

Example (Visual Basic .Net)

```
' Set the offset value of the WE7111 module to 1.24 V.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
Dim data As Double
data = 1.24
ret = Module1.SetControlEx ("Offset", data)
```

GetControlEx

Description

Gets the current values of the individual control Parameters of the module. You can get the control parameters by specifying the data type.

Syntax

GetControlEx (*command* As String, ByRef param As Byte) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As Byte) As Short
GetControlEx (*command* As String, ByRef param As SByte) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As SByte) As Short
GetControlEx (*command* As String, ByRef param As UInt16) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As UInt16) As Short
GetControlEx (*command* As String, ByRef param As Short) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As Short) As Short
GetControlEx (*command* As String, ByRef param As UInt32) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As UInt32) As Short
GetControlEx (*command* As String, ByRef param As Integer) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As Integer) As Short
GetControlEx (*command* As String, ByRef param As Single) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As Single) As Short
GetControlEx (*command* As String, ByRef param As Double) As Short
GetControlEx (*command* As String, *size* as Integer, ByRef *param*() As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN)	ASCII command name that is defined for each module
size (IN)	Number of data points to be retrieved
param (OUT)	Data pointer

Example (Visual Basic .Net)

```
' Get the offset value of the WE7111 module.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
Dim data As Double
ret = Module1.GetControlEx ("Offset",  data)
```

SetQueryControl

Description

Sets the control parameter using the module handle and command, and then gets the current value of the control parameter.

Syntax

SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As SByte, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As SByte, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As Byte, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As Byte, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As Short, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As Short, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As UInt16, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As UInt16, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As Integer, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As Integer, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As UInt32, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As UInt32, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As Single, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As Single, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*As Double, *rtype* As Short, ByRef *getParam* As Object) As Short
SetQueryControl (*command* As String, *paramNum* As Integer,ByRef *setParam*() As Double, *rtype* As Short, ByRef *getParam* As Object) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN)	ASCII command name that is defined for each module
paramNum (IN)	Number of data to send
setParam (IN)	Pointer to the data to send

rtype (IN) Data type of the data to receive
The following symbols are defined:

```
WE_NULL       ' No parameter
WE_UBYTE      ' 8-bit unsigned integer
WE_SBYTE      ' 8-bit signed integer
WE_UWORD      ' 16-bit unsigned integer
WE_SWORD      ' 16-bit signed integer
WE_ULONG      ' 32-bit unsigned integer
WE_SLONG      ' 32-bit signed integer
WE_FLOAT      ' 32-bit real number
WE_DOUBLE     ' 64-bit real number
```

getParam (OUT) Pointer to the data to receive

Note:

Use this class only with the WE7021 module.

Example (Visual Basic .Net)

```
' Using the WE7021 module, retrieve the block data from the
' connected device.
Dim pBuf As Object
Dim rSize As Long
rSize = 2002
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7021 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7021", 1)
ret = Module1.SetQueryControl ("DATA", 1, rSize, WeControl.WE_UWORD, pBuf)
```

SetScaleInfo

Description

Sets scale conversion information to the measurement module. This function is equivalent to the settings in the convert scale dialog box. The information is stored to the module through operations such as update preset.

Syntax

SetScaleInfo (*ch* As Integer, *LSInfoNum* As Integer, *LSInfo*() As LinearScaleInfo) As Short

Parameters

ch (IN) Channel number
One origin. If the modules are linked, it is the link number. -1 represents all channels.

LSInfoNum (IN) Number of LinearScaleInfo

LSInfo (IN) Scale conversion table information
If -1 is specified for ch, the number of LSInfo Parameters that needs to be specified is equal to the number of channels.

```
Structure LinearScaleInfo       ' Whether or not to enable scaling.
    scaling As Integer          ' (1: enable scaling, 0: disable scaling)
    rsrv  As Integer            ' Reserved.
    a As Double                 ' The value a of scaling parameter ax+b.
    b As Double                 ' The value b of scaling parameter ax+b.
    label As String             ' Label name (32 characters).
    unit As String              ' Name of the unit (16 characters).
End Structure
```

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7275 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7275:1", 1)
Dim info(WeControl.LinearScaleInfoNum-1) As LinearScaleInfo
info.list(0).scaling = 1
info.list(0).a = 1.0
info.list(0).b = 0.0
info.list(0).label = "CH1" + Chr$(0)
info.list(0).unit = "unit1" + Chr$(0)
info.list(1).scaling = 1
info.list(1).a = 2.0
info.list(1).b = 0.0
info.list(1).label = "CH2" + Chr$(0)
info.list(1).unit = "unit2" + Chr$(0)
' Set the scale information of all channels (2 channels).
ret = Module1.SetScaleInfo (-1, WeControl.LinearScaleInfoNum, info)
```

GetScaleInfo

Description

Gets scale conversion information of the measurement module.

Syntax

GetScaleInfo (*ch* As Integer, *LsInfoNum* As Integer *LSInfo*() As LinearScaleInfo) As Short

Parameters

ch (IN) Channel number
One origin. If the modules are linked, it is the link number. -1 represents all channels.

LSInfoNum (IN) Number of LinearScaleInfo

LSInfo (IN) Scale conversion table information
If -1 is specified for ch, the number of info Parameters that needs to be specified is equal to the number of channels.

```
Structure LinearScaleInfo
    scaling As Long      ' Whether or not to enable scaling.
    rsrv  As Long        ' (1: enable scaling, 0: disable scaling)
    a As Double          ' Reserved.
    b As Double          ' The value a of scaling parameter ax+b.
    label As String      ' The value b of scaling parameter ax+b.
    unit As String       ' Label name (32 characters).
End Structure            ' Name of the unit (16 characters).
```

Example (Visual Basic .Net)

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7275 module with a link number of 1.
ret = Module1.OpenModule (Station.hSt, "WE7275:1", 1)
Dim info(WeControl.LinearScaleInfoNum-1) As LinearScaleInfo
' Queries the scale information of all channels (2 channels).
ret = Module1.GetScaleInfo (-1, WeControl.LinearScaleInfoNum, info)
```

SetModuleBus

Description

Sets the trigger source/time base source (sampling clock) and the input/output setting of the arming signal of the module. If a module is specified that does not have a trigger source, time base source, or an arming signal input/output function, the settings are discarded.

Syntax

SetModuleBus (*InItem* As Byte, *OutItem* As Byte, *InClock* As Byte, *ArmItem* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

InItem (IN)	Trigger input selection	
	WE_TRGNONE	' Do not use the bus trigger signal.
	WE_TRG1	' Use bus trigger signal “BUSTRG 1.”
	WE_TRG2	' Use bus trigger signal “BUSTRG 2.”
OutItem (IN)	Trigger output selection	
	WE_TRGNONE	' Do not output to the trigger bus signal.
	WE_TRG1	' Output to bus trigger signal “BUSTRG 1.”
	WE_TRG2	' Output to bus trigger signal “BUSTRG 2.”
	WE_BOTH	' Output to both bus trigger signals “BUSTRG 1, 2.”
InClock (IN)	Sampling clock input selection	
	WE_CMNCLKNONE	' Do not use the bus clock signal.
	WE_CMNCLK	' Use the sampling clock signal.
ArmItem (IN)	Arming signal input selection	
	WE_ARMNONE	' Do not use arming signal.
	WE_ARM	' Use arming signal.

Example (Visual Basic .Net)

```
' Use bus trigger signal "BUSTRG 2", output bus trigger signal
' "BUSTRG 1" and "BUSTRG 2", do not use the sampling clock signal, use
' arming signal.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
ret = Module1.SetModuleBus (WeControl.WE_TRG2, WeControl.WE_BOTH,
WeControl.WE_CMNCLKNONE,WeControl.WE_ARM)
```

GetModuleBus

Description

Gets the trigger source/time base source (sampling clock) and the input/output setting of the arming signal of the module. If a module is specified that does not have a trigger source, time base source, or an arming signal input/output function, a “0” is returned for each setting.

Syntax

GetModuleBus (ByRef *InItem* As Byte, ByRef *OutItem* As Byte, ByRef *InClock* As Byte, By Ref *ArmItem* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

InItem (OUT)	Trigger input setting	
	WE_TRGNONE	' Do not use the bus trigger signal.
	WE_TRG1	' Use bus trigger signal “BUSTRG 1.”
	WE_TRG2	' Use bus trigger signal “BUSTRG 2.”
OutItem (OUT)	Trigger output setting	
	WE_TRGNONE	' Do not output to the trigger bus signal.
	WE_TRG1	' Output to bus trigger signal “BUSTRG 1.”
	WE_TRG2	' Output to bus trigger signal “BUSTRG 1.”
	WE_BOTH	' Output to both bus trigger signals “BUSTRG 1, 2.”
InClock (OUT)	Sampling clock input setting	
	WE_CMNCLKNONE	' Do not use the sampling clock signal.
	WE_CMNCLK	' Use the sampling clock signal.
ArmItem (OUT)	Arming signal input setting	
	WE_ARMNONE	' Do not use arming signal.
	WE_ARM	' Use arming signal.

Example (Visual Basic .Net)

```
' Gets the trigger/time base source and arming settings of the module.
Dim InItem As Byte
Dim OutItem As Byte
Dim InClock As Byte
Dim ArmItem As Byte
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the first WE7111 module with a link number of 3.
ret = Module1.OpenModule (Station, "WE7111:1", 3)
ret = Module1.GetModuleBus (InItem, OutItem, InClock, ArmItem)
```

ShowModuleWindow

Description

Displays the module GUI Window for controlling the module.

Syntax

ShowModuleWindow (ByRef *hWnd* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

hWnd (OUT)	GUI window handle. This method is available to use when changing the position of the GUI window.

Note:

The operation panel that is displayed using this method can be used only to make setting changes. It cannot be used to start/stop the module. In addition, for operation panels that display measured values (instantaneous values), the displayed values are not updated.

Example (Visual Basic .Net)

```
' Open the WE7111 GUI panel.
Dim hWnd As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with  a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.ShowModuleWindow (hWnd)
' Move the window position to x = 520 and y = 220.
ret = SetWindowPos (hWnd, 0, 520, 220, 0, 0, 1)
```

CloseModuleWindow

Description

Closes the module GUI Window used to control the module.

Syntax

CloseModuleWindow () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Close the WE7111 GUI panel.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.CloseModuleWindow ()
```

IsModuleWindow

Description

Queries whether or not the module GUI window for controlling the module is open.

Syntax

IsModuleWindow (ByRef *sw* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

sw (OUT)	Returns 1 if the module window is open, 0 if it is not.

Example (Visual Basic .Net)

```
' Query whether or not the WE7111 GUI panel is open.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
Dim sw As Byte
ret = Module1.IsModuleWIndow (sw)
```

ShowLinearScaleWindow

Description

Displays the convert scale dialog box.

Syntax

ShowLinearScaleWindow (ByRef *hWnd* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

hWnd (OUT)	Convert scale dialog box handle Used when changing the position of the dialog box, for example.

Example (Visual Basic .Net)

```
' Display the convert scale dialog box for the WE7251 module.
Dim hWnd As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 2.
ret = Module1.OpenModule (Station, "WE7251:1", 2)
ret = Module1.ShowLinearScaleWindow (hWnd)
' Calls the WIN32API and moves the window position to x = 520, y = 220.
ret = SetWindowPos(hWnd, 0, 520, 220, 0, 0, 1)
```

CloseLinearScaleWindow

Description

Closes the convert scale dialog box.

Syntax

CloseLinearScaleWindow () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Close the convert scale dialog box for the WE7251 module.
Dim hWnd As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 2.
ret = Module1.OpenModule (Station, "WE7251:1", 2)
ret = Module1.ShowLinearScaleWindow (hWnd)
ret = Module1.CloseLinearScaleWindow ()
```

IsLinearScaleWindow

Description

Queries whether or not the convert scale dialog box is open.

Syntax

IsLinearScaleWindow (ByRef *sw* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

sw (OUT) Returns 1 if the module operation panel is displayed, 0 if it is not.

Example (Visual Basic .Net)

```
' Query whether or not the convert scale dialog box for the WE7251 module
' is open.
Dim hWnd As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 2.
ret = Module1.OpenModule (Station, "WE7251:1", 2)
Dim sw As Byte
ret =Module1.IsLinearScaleWindow(sw)
```

Start

Description

Starts the operation of the measurement module.

Syntax

Start () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Note:

This method only issues the start command to a module. For example,it does not carry out operations such as synchronize to the end of an acquisition on the acquisition module. If you need such termination process, use the IsRun method, which polls (monitors) the end of the execution.

Example (Visual Basic .Net)

```
' Start WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.Start ()
```

Stop

Description

Stops the operation of the measurement module.

Syntax

Stop () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Stop WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.Start ()
  ......
ret = Module1.Stop ()
```

StartEx

Description

Starts the operation of the measurement module. This function can be used to simply issue the start command (same as Start ()) or or have the module notify the end of the acquisition with an event by specifying the operation mode. In the event notification mode, event notification for each block and event notification according to the logic block during the free run mode are possible.

Syntax

StartEx (*blockLen* As Integer, *blockCount* As Byte, *acqCount* As Integer, *mode* As Short) As Short

Old interface

StartEx (*blockLen* As Integer, *blockCount* As Byte, *acqCount* As Integer, *mode* As Short, ByRef *evHandle* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

blockLen (IN)	Number of data points per block (record length)
blockCount (IN)	Number of memory partitions (blocks) Number of blocks can only be specified in powers of two's. Make sure the following equation is satisfied: "blockLen* (2 to the blockCount power) ≤ Acquisition memory length in the module."
acqCount (IN)	Number of acquisitions. The data acquisition operation terminates after acquiring the amount of data specified by this number. If 0 is specified, the operation continues until the user issues the stop command.
mode (IN)	Acuquisition operation mode WE_ACQ_NO_EVENT ' Do not generate events WE_ACQ_BLOCK_EVENT ' Generate events for each block WE_ACQ_STOP_EVENT ' Generate events after acquiring all ' blocks and the operation stops
evHandle (OUT)	Even handle (old interface) Also used in StopEx(). (The new interface stores the event handle within the class and releases the event using the value in StopEx().)

Note:

acqCount is ignored when WE_ACQ_BLOCK_EVENT is specified. In other words, data acquisition operation continues until the stop command is issued. When the acquisition mode is set to free run, blockCount is ignored.

Example (Visual Basic .Net)

- **Example in which data are read using the event notification in the trigger mode on the WE7251**

```
' Initialization procedure
' Initialize
ret = Comm.Init(WeEvent1.hwnd, "Optical devicename=we7036", WE_CONTROLLER)
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation("Station1")
' Turn ON Station 1.
ret = Station.Power(WeControl.WE_ON)
' Open module WE7251 with a link number of 1.
ret = Module1.OpenModule(Station, "WE7251:1", 1)
' Enable aquisition on CH1.
ret = Module1.SetControl("CH1:On", "On")
' Set the aquisition mode to "Triggered."
ret = Module1.SetControl("Acquisition Mode", "Triggered")
  ......
```

```
' Start procedure
' Set the sampling interval to 0.001 s.
ret = Module1.SetControl("Sampling Interval", "0.001")
' Set the number of memory partitions to 2 (2 to the 1st power)
' and the number of measurements to 2.
' Request that the WeEvent be issued when two blocks of data
' (1000 points) are acquired.
' Therefore, an event is notified after 2 s (sampling interval of
' 0.001 x block length of 1000 x number of measurments of 2).
' Start measurement.
ret = Module1.StartEx(1000, 1, 2, WeControl.WE_ACQ_STOP_EVENT)
  ......

' WeEvent handler
Dim recSize As Long
Dim buf() As Double
Dim sparam As ScalingParam
If ev = WeContorol.WE_EV_MEASEND Then ' If it is an acquisition stop event,
  sparam.a = 1
  sparam.b = 0
recSize = 1000*8
' Allocate a buffer for 1000 points.
ReDim buf(999) As Double
' Read the data from CH1 block 0.
' Read the voltage.
ret = Module1.GetScaleData(1, 0, sparam, recSize, WeControl.WE_DOUBLE,
buf(0))
' Display the data of block 0 or make an analysis.
  ......
' Read the data from CH1 block 1.
' Read the voltage.
ret = Module1.GetScaleData(1, 1, sparam, recSize, WeControl.WE_DOUBLE,
buf(0))
  ......
End If

' Stop procedure
' Stops the measurement and releases the event.
ret = Module1.StopEx()
  ......
```

- **Example in which data are read using the event notification in the free run mode on the WE7251**

```
' Initialization procedure
' Initialize
ret = Comm.Init(WeEvent1.hwnd, "Optical devicename=we7036", WE_CONTROLLER)
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation("Station1")
' Turn ON Station 1.
ret = Station.Power(WeControl.WE_ON)
' Open module WE7251 with a link number of 1.
ret = Module1.OpenModule(Station,"WE7251:1", 1)
' Enable aquisition on CH1.
ret = Module1.SetControl("CH1:On", "On")
' Set the aquisition mode to "Free Run."
ret = Module1.SetControl("Acquisition Mode", "Free Run")
  ......
```

```
' Start procedure
' Set the sampling interval to 0.01 s.
ret = Module1.SetControl("Sampling Interval", "0.01")
' Request that the WeBlock event be issued when 100 points of
' data are acquired.
' Therefore, an event is notified every 1 s (sampling interval of
' 0.01 x block length of 100).
' In the free run mode, the number of blocks is ignored and the
' number of acquisitions is set to infinity.
ret = Module1.StartEx(100, 0, 0, WeControl.WE_ACQ_BLOCK_EVENT)
' Start measurement.
  ......

' WeBlock handler
Dim recSize As Long
Dim buf () As Double
Dim sparam As ScalingParam
  sparam.a = 1
  sparam.b = 0
' Since only the CH1 data are read, the buffer size is 100 points
' x 8 bytes (Double type).
recSize = 100*8
ReDim buf (00) As Double ' Allocate a buffer for 100 points.
' Read the voltage.
ret = Module1.GetScaleData(1, blockNo, sparam, recSize, WeControl.WE_DOUBLE,
buf(0))
' Display the data or make an analysis.
  ......

' Stop procedure
' Stops the measurement and releases the event.
ret = Module1.StopEx()
  ......
```

StopEx

Description

Stops the operation of the measurement module. Use this method to stop the operation only if the acquisition was started with the StartEx method.

Syntax

StopEx () As Short

Old interface

StopEx (*evHandle* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Be sure to execute StopEx(), if you execute StartEx().

old interface

evHandle (IN)	Even handle to be released Specify the event handle obtained when StartEx() was executed.

Example (Visual Basic .Net)

```
ret = Module1.StartEx(1000, 0, 0, WE_ACQ_BLOCK_EVENT)
  ......
ret = Module1.StopEx()
```

IsRun

Description

Queries the execution state of the measurement module (run/stop).

Syntax

IsRun (ByRef *status* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

status (OUT) Run state (WE_RUN) or stop state (WE_STOP)

Example (Visual Basic .Net)

```
' Queries the execution state of WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.Start ()
Dim status As Byte
Do
ret = Module1.IsRun (status)
If status=0 Then
Exit Do
End If
Loop
```

GetAcqDataInfoEx

Description

Reads the acquisition data information of the measurement module.

Syntax

GetAcqDataInfo(*ch* As Short, *blockNo* As Integer, *info*() As AcqDataInfoEx2, ByRef *infoNum* As Short) As Short

Old interface

GetAcqDataInfo(*ch* As Short, *blockNo* As Integer, ByRef *info* As AcqDataInfo, ByRef *infoNum* As Short) As Short

GetAcqDataInfoEx(*ch* As Short, *blockNo* As Integer, ByRef *info* As AcqDataInfoEx, ByRef *infoNum* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

An error occurs if you specify a block number that exceeds the valid number of data blocks.

Parameters

ch (IN) Channel number.
One origin. If the modules are linked, it is the link number. -1 represents all channels.

blockNo (IN) Block number
For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.

info (OUT) Data information pointer
Structure AcqDataInfoEx2

```
channel As ushort          // Channel number (1 or greater)
dataType As ushort         // Acquisition data type
blockNum As ushort         // Number of valid blocks
startBit As ushort         // Valid bit start position for integers
                           // (bit0 orgreater)
effectiveBit As ushort     // Valid bit length for integers (0 specifies up to
                           // the highest bit)
trigActive As ushort       // Trigger active (0/1)
record As uint             // Record length (number of samples)
recordLen As uint          // Display record length
trigPosition As uint       // Trigger position
                           // (record position of the 0 base point)
interval As double         // Sampling interval (s)
vResolution As double      // Scaling factor for converting physical values
vOffset As double          // Offset for converting physical values
trigLevel As double        // Trigger level (the order of the physical value)
trigWidth As double        // Trigger width (the order of the physical value)
plusOverData As double     // Positive overrange value
                           // (the order of the acquisition data)
minusOverData As double    // Negative overrange value
                           // (the order of the acquisition data)
nonData As double          // Illegal value (the order of the acquisition data)
dispMaxData As double      // Maximum display value
                           // (the order of the acquisition data)
dispMinData As double      // Minimum display value
                           // (the order of the acquisition data)
date As string             // Date of acquisition
time As string             // Time of acquisition
vUnit As string            // Physical value unit
xUnit As string            // Horizontal axis unit
```

info (OUT) Data information pointer (for old interface)
Structure AcqDataInfo

```
channel As ushort          // Channel number(1 or greater)
dataType As ushort         // Acquisition data type
blockNum As ushort         // Number of valid blocks
startBit As ushort         // Valid bit start position for integers
                           // (bit0 or greater)
effectiveBit As ushort     // Valid bit length for integers
                           // (0 specifies up to the highest bit)
trigActive As ushort       // Trigger active (0/1)
record As uint             // Record length (number of samples)
recordLen As uint          // Display record length
trigPosition As uint       // Trigger position
                           // (record position of the 0 base point)
interval As double         // Sampling interval (s)
```

```
    vResolution As double       // Scaling factor for converting physical values
    vOffset As double           // Offset for converting physical values
    trigLevel As double         // Trigger level (the order of the physical value)
    trigWidth As double         // Trigger width (the order of the physical value)
    plusOverData As double      // Positive overrange value
                                // (the order of the acquisition data)
    minusOverData As double     // Negative overrange value
                                // (the order of the acquisition data)
    nonData As double           // Illegal value (the order of the acquisition data)
    dispMaxData As double       // Maximum display value
                                // (the order of the acquisition data)
    dispMinData As double       // Minimum display value
                                // (the order of the acquisition data)
End Structure
```

info (OUT) Data information pointer (old interface)

```
Structure AcqDataInfoEx
    channel As ushort           // Channel number(1 or greater)
    dataType As ushort          // Acquisition data type
    blockNum As ushort          // Number of valid blocks
    startBit As ushort          // Valid bit start position for integers
                                // (bit0 or greater)
    effectiveBit As ushort      // Valid bit length for integers
                                // (0 specifies up to the highest bit)
    trigActive As ushort        // Trigger active (0/1)
    record As uint              // Record length (number of samples)
    recordLen As uint           // Display record length
    trigPosition As uint        // Trigger position
                                // (record position of the 0 base point)
    time As int                 // Acquisition time (time_t format)        old rsv6
    interval As double          // Sampling interval (s)
    vResolution As double       // Scaling factor for converting physical values
    vOffset As double           // Offset for converting physical values
    trigLevel As double         // Trigger level (the order of the physical value)
    trigWidth As double         // Trigger width (the order of the physical value)
    plusOverData As double      // Positive overrange value
                                // (the order of the acquisition data)
    minusOverData As double     // Negative overrange value
                                // (the order of the acquisition data)
    nonData As double           // Illegal value (the order of the acquisition data)
    dispMaxData As double       // Maximum display value
                                // (the order of the acquisition data)
    dispMinData As double       // Minimum display value
                                // (the order of the acquisition data)
End Structure

    dataType
    ' The data type of acquisition is prescribed by the following constants.
    ' WE_NULL          No parameter
    ' WE_UBYTE         Unsigned 8-bit integer
    ' WE_SBYTE         Signed 8-bit integer
    ' WE_UWORD         Unsigned 16-bit integer
    ' WE_SWORD         Signed 16-bit integer
    ' WE_ULONG         Unsigned 32-bit integer
    ' WE_SLONG         Signed 32-bit integer
```

```
' WE_FLOAT          32-bit real number
' WE_DOUBLE         64-bit real number
```

infoNum (OUT) Number of AcqDataInfo that is actually read

Note:

When the acquisition mode is set to free run, the block number, blockNum, is always “1.”
The information attached to the acquired data varies depending on the module as follows.

- **WE7081(Acquisition Mode)**

Description:Information regarding raw data
dataType:Depends on the channel definition
startBit: Depends on the channel definition
effectiveBit: Depends on the channel definition
trigActive: When the trigger type is not Off: 1
When set to other or BUSTRG: 0
record:Record length setting
recordLen:Record-1
trigPosition:Pre-trigger setting
interval:Sampling interval setting
vResolution:1.0
vOffset:0.0
trigLevel:Trigger level setting
trigWidth:Meaningless
plusOverData:Meaningless
minusOverData:Meaningless
nonData:Meaningless
dispMaxData:Meaningless
dispMinData:Meaningless

- **WE7111**

Description: Information regarding the raw data
dataType: WE_SWORD in the Average mode, WE_SBYTE for all other modes.
blockNum: 1
startBit: Meaningless
effectiveBit: Meaningless
trigActive: 1 for data from the channel which is the trigger source, 0 otherwise. Meaningless when the trigger source is set to BUSTRG.
record: Record length setting value
recordLen: Record length, depends on the time/div setting.
trigPosition: Trigger position, depends on the trigger position setting.
interval: Depends on the time/div setting.
vResolution: Depends on the V/div and probe attenuation settings.
vOffset: Depends on the offset voltage and probe attenuation settings
trigLevel: Depends on the trigger level and probe attenuation settings.
trigWidth: Meaningless
plusOverData: 0x3F00 in the Average mode, 0xFE for all other modes.
minusOverData: 0xC000 in the Average mode, 0x01 for all other modes
nonData: 0x8000 in the Average mode, 0xFE for all other modes.
dispMaxData: 0x3E00 in the Average mode, 0xFD for all other modes.
dispMinData: 0xC100 in the Average mode, 0x02 for all other modes

- **WE7116**

Description: Information regarding A/D raw data
DataType: WE_SWORD
blockNum: Number of memory partitions
StartBit: Meaningless.
EffectiveBit: Meaningless.
TrigActive: When the trigger type is not Off: 1
When set to other or BUSTRG: 0
Record: Record length setting
RecordLen: Record -1
TrigPosition: Pre-trigger setting
Interval: Sampling interval setting
VResolution: Scaling factor for converting physical values
VOffset: Offset for converting physical values
TrigLevel: Trigger level setting
TrigWidth: Meaningless.
PlusOverData: DispMaxData+1
MinusOverData: DispMinData-1
NonData: PlusOverData
DispMaxData: Maximum display value
DsipMinData: Minimum display value

- **WE7141**

Description: Information regarding the physical value (depends on the measurement function setting)
dataType: WE_DOUBLE
blockNum: 1
startBit: Meaningless
effectiveBit: Meaningless
trigActive: Meaningless
record: Record length setting value
recordLen: Meaningless
trigPosition: Meaningless
interval: Sampling interval setting
vResolution: 1.0
vOffset: 0.0
trigLevel: Meaningless
trigWidth: Meaningless
plusOverData: +∞
minusOverData: -∞
nonData: Non number (NAN)
dispMaxData: Depends on the measurement function setting
dispMinData: Depends on the measurement function setting

- **WE7231**

Description: Information regarding the physical value (temperature, voltage, or resistance)
DataType: WE_FLOAT
BlockNum: Number of blocks measured after starting data acquisition
StartBit: Meaningless
EffectiveBit: Meaningless
TrigActive: Meaningless
Record: Record length setting
RecordLen: Meaningless
TrigPosition: Meaningless
Interval: Depends on the Sampling Interval setting
Vresolution: 1.0

Voffset: 0.0
TrigLevel: Meaningless
TrigWidth: Meaningless
PlusOverData: DispMaxData +100
MinusOverData: DispMinData -100
NonData: Non number (NAN)
DispMaxData: Depends on the measurement range
DsipMinData: Depends on the measurement range

- **WE7235**

Description: Information regarding A/D raw data
DataType: WE_SWORD
blockNum: Number of memory partitions
StartBit: Meaningless
EffectiveBit: Meaningless
TrigActive: When the trigger type is not Off: 1
When set to other or BUSTRG: 0
Record: Record length setting
RecordLen: Record -1
TrigPosition: Pre-trigger setting
Interval: Sampling interval setting
VResolution: Scaling factor for converting physical values
VOffset: Offset for converting physical values
TrigLevel: Trigger level setting
TrigWidth: Meaningless
PlusOverData: DispMaxData +1
MinusOverData: DispMinData -1
NonData: PlusOverData
DispMaxData: Maximum display value
DsipMinData: Minimum display value

- **WE7241**

Description: Information regarding the physical value (temperature or voltage)
dataType: WE_FLOAT
blockNum: 1
startBit: Meaningless
effectiveBit: Meaningless
trigActive: Meaningless
record: Record length setting
recordLen: Meaningless
trigPosition: Meaningless
interval: Sampling interval setting
vResolution: 1.0
vOffset: 0.0
trigLevel: Meaningless
trigWidth: Meaningless
plusOverData: “dispMaxData” +100
minusOverData: “dispMinData” -100
nonData: Non number (NAN)
dispMaxData: Depends on the measurement range setting
dispMinData: Depends on the measurement range setting

- **WE7245**

Description: Information regarding the A/D raw data
DataType: WE_SWORD
blockNum: Number of memory partitions
StartBit: Meaningless
EffectiveBit: Meaningless
TrigActive: 1 if the trigger type is not Off, 0 otherwise. 0 also for BUSTRG.
Record: Record length setting
RecordLen: Record-1
TrigPosition: Pre-trigger setting
Interval: Sampling interval setting
VResolution: Depends on the measurement range, the calibration at the time of shipment, and the execution of balancing.
VOffset: Depends on the measurement range, the calibration at the time of shipment, and the execution of balancing.
TrigLevel: Trigger level setting
TrigWidth: Meaningless
PlusOverData: DispMaxData+1
MinusOverData: DispMinData-1
NonData: PlusOverData
DispMaxData: Depends on the measurement range and the calibration at the time of shipment.
DsipMinData: Depends on the measurement range and the calibration at the time of shipment.

- **WE7251**

Description: Information regarding raw data
dataType: WE_SWORD
blockNum: Number of memory partitions
startBit: Meaningless
effectiveBit: Meaningless
trigActive: 1 if the trigger type is not Off 0 otherwise. 0 also for BUSTRG.
record: Record length setting
recordLen: “record” - 1
trigPosition: Pre-trigger setting
interval: Sampling interval setting
vResolution: Measurement range, depends on the calibration value at the time of shipment
vOffset: Measurement range, depends on the calibration value at the time of shipment
trigLevel: Trigger level setting
trigWidth: Difference between the High and Low levels when the trigger type is In or OUT, meaningless for all other types or when the trigger source is set to BUSTRG.
plusOverData: “dispMaxData” + 1
minusOverData: “dispMinData” - 1
nonData: “plusOverData”
dispMaxData: Measurement range, depends on the calibration value at the time of shipment
dispMinData: Measurement range, depends on the calibration value at the time of shipment

- **WE7261/WE7262**

Description: Information regarding the In/Out bit data
dataType: BIT16_TYP
blockNum: 1
startBit: 0
effectiveBit: 15
trigActive: Meaningless
record: 8192
recordLen: Meaningless
trigPosition: Meaningless
interval: Sampling interval setting
vResolution: Meaningless
vOffset: Meaningless
trigLevel: Meaningless
trigWidth: Meaningless
plusOverData: Meaningless
minusOverData: Meaningless
nonData: Meaningless
dispMaxData: 2
dispMinData: -1

- **WE7271/7272/7275**

Description: Information regarding raw data
dataType: WE_SWORD
blockNum: Number of memory partitions
startBit: Meaningless
effectiveBit: Meaningless
trigActive: 0 when the trigger type is set to OFF, 1 otherwise. Meaningless when the trigger source is set to BUSTRG.
record: Record length setting
recordLen: “record” - 1
trigPosition: Pre-trigger setting
interval: Sampling interval setting
vResolution: Measurement range, depends on the calibration value at the time of shipment
vOffset: Measurement range, depends on the calibration value at the time of shipment
trigLevel: Trigger level setting
trigWidth: Meaningless
plusOverData: “dispMaxData” + 1
minusOverData: “dispMinData” - 1
nonData: “plusOverData”
dispMaxData: Measurement range, depends on the calibration value at the time of shipment
dispMinData: Measurement range, depends on the calibration value at the time of shipment

- **WE7311**

Description: Information regarding raw data
DataType: WE_SBYTE
BlockNum: Specified number of blocks
StartBit: Meaningless
EffectiveBit: Meaningless
TrigActive: 1 for data from the channel which is the trigger source. Meaningless for all other channels or when the trigger source is set to BUSTRG/External 1M/50.
Record: Record length setting
RecordLen: Depends on the record length and Time/div settings
TrigPosition: Depends on the pre-trigger, record length, and trigger position settings
Interval: Depends on the sampling interval and Time/div settings
Vresolution: Depends on the Range, V/div, and probe attenuation settings
Voffset: Depends on the offset voltage and probe attenuation settings
TrigLevel: Depends on the trigger level and probe attenuation settings
TrigWidth: Meaningless
PlusOverData: 0x7F
MinusOverData: 0x81
NonData: 0x80
DispMaxData: 0x7D
DsipMinData: 0x83

- **WE7521**

For Counter Mode
Description: Information regarding raw data
DataType: When the measurement function is UpDown1/UpDown2/UpDown4:
WE_SLONG
When the measurement function is Frequency:
Prior to transformation by the TransAcqData method: WE_ULONG
After transformation by the TransAcqData method: WE_FLOAT
When the measurement function is other than above: WE_ULONG
blockNum: Number of memory partitions
StartBit: Meaningless
EffectiveBit:
When the measurement function is Period/TI: value depending on the period stop determination time
When the measurement function is Frequency: value depending on the 128+period stop determination time
When the measurement function is other than above: 0
WE Control API Draft Version 1/3
TrigActive: 1 if the trigger type is not OFF, 0 otherwise. 0 also for BUSTRG.
Record: Record length setting
RecordLen: Record –1
TrigPosition: Pre-trigger setting
Interval: Sampling interval setting
VResolution: Scaling factor for converting physical values
VOffset: Offset for converting physical values
TrigLevel: Trigger level setting
TrigWidth: Meaningless
PlusOverData: DispMaxData + 1
MinusOverData: DispMinData – 1
NonData: When the measurement function is UpDown1/UpDown2/UpDown4: 0x80000000
When the measurement function is other than above: 0xffffffff
DispMaxData: Maximum display value
DsipMinData: Minimum display value

For time stamp mode
Description: Information regarding raw data
DataType: WE_ULONG
  Upper 24 bits: Time stamp data Lower 8 bits: Input change information
blockNum: Meaningless
StartBit: 8
EffectiveBit: 24
TrigActive: Meaningless
Record: Record length retrieved between latches
RecordLen: Meaningless
TrigPosition: Meaningless
Interval: Meaningless
VResolution: Scaling factor for converting physical values
VOffset: Offset for converting physical values
TrigLevel: Meaningless
TrigWidth: Meaningless
PlusOverData: Meaningless
MinusOverData: Meaningless
NonData: Meaningless
DispMaxData: Meaningless
DsipMinData: Meaningless

Example (Visual Basic .Net)

```
' Query the additional information pertaining to the acquisition data of
' all channels of the WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim info (2) As AcqDataInfoEx2
Dim num As Short
ret = Module1.GetAcqDataInfo (-1, 0, info, num)
```

GetAcqDataSize

Description

Determines the acquisition data size (number of bytes) of the measurement module. The number of data bytes are adjusted according to the data type.

Syntax

GetAcqDataSize (*ch* As Short, *blockNo* As Integer, ByRef *pointNum* As Integer, ByRef *dataSize* As Integer, ByRef *ptype* As Short, ByRef *chNum* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
blockNo (IN)	Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.
pointNum (OUT)	Number of data points. If -1 is specified for ch, the number of data points, of all channels that can currently acquire data, is returned.
dataSize (OUT)	Total number of bytes. If -1 is specified for ch, the data size, of all channels that can currently acquire data, is returned. This number is undefined for free run acquisition mode.
ptype (OUT)	Data type
chNum (OUT)	Total number of channels that can currently acuire data

Note:

For the free run acquisition mode, call the latch method LatchData as necessary before calling this method. The data size between latches can be obtained. For details, see *chapter 4, "Data Acquisition Model" in the WE Control API User's Manual (IM707741-61E)*.

Example (Visual Basic .Net)

```
' Get the data size of one channel of the WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim pointNum As Integer
Dim dataSize As Integer
DIm ptype As Short
Dim chNum As Short
ret = Module1.GetAcqDataSize(1, 0, pointNum, dataSize, ptype, chNum)
```

GetAcqData

Description

Retrieves acquisition data from the measurement module. The retrieved data are raw data that have been A/D converted. The data format depends on the measurement module. The relevant information can be obtained with the GetAcqDataInfo method.

Syntax

GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As SByte) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Byte) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt16) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Integer) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt32) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt64) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Old interface

GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As SByte, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Byte, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Short, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt16, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Integer, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt32, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As UInt64, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Single, ByRef *ptype* As Short) As Short
GetAcqData (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Double, ByRef *ptype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN) — Channel number. One origin is. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.

blockNo (IN) — Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.

recSize (IN/OUT) — Data buffer size for retrieving data or the data size that was retrieved (bytes)

buf (OUT) — Pointer to the data buffer. Allocate enough buffer space to store the acquisition data being retrieved. You can use the GetAcqDataSize () to query the data size.

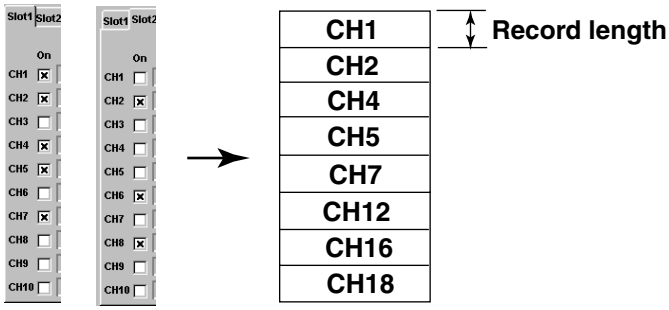
ptype (OUT) — Data type (old interface)

```
WE_NULL      ' No parameter
WE_UBYTE     ' 8-bit unsigned integer
WE_SBYTE     ' 8-bit signed integer
WE_BIT8      ' 8-bit logical value
WE_UWORD     ' 16-bit unsigned integer
WE_SWORD     ' 16-bit signed integer
WE_BIT16     ' 16-bit logical value
WE_ULONG     ' 32-bit unsigned integer
WE_SLONG     ' 32-bit signed integer
WE_BIT32     ' 32-bit logical value
WE_FLOAT     ' 32-bit real number
WE_DOUBLE    ' 64-bit real number
WE_BIT64     ' 64-bit logical value
```

Note:

For the free run acquisition mode, call the latch method LatchData () before calling this method as necessary. The data size between latches can be obtained. For details, see *chapter 4, "Data Acquisition Model" in the WE Control API User's Manual (IM707741-61E)*.

The following figure depicts the data format when "-1", that represents all channels, is specified for the channel number (ch (IN)). If two modules are linked and the channels are enabled as shown in the lower left figure, the disabled channels are skipped and data are continuously read as shown in the lower right figure.



Example (Visual Basic .Net)

- **Retrieving the raw data of channel 1 of WE7111**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
  ..... ' Waiting for completion of data acquisition after starting
measurement
Dim recSize As Integer
Dim ptype As Short
' Allocate a buffer for a memory length of 10000 points.
Dim buf (10000) As Byte
' The number of bytes of the buffer is 10000 bytes
recSize = 10000
ret = module1.GetAcqData (1, 0, recSize, buf(0), ptype)
```

- **Retrieving the raw data for all channels of the WE7251**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 2.
ret = Module1.OpenModule (Station, "WE7251:1", 2)
  ..... ' Waiting for completion of data acquisition after starting
measurement
Dim recSize As Integer
Dim ptype As Short
' Allocate a buffer for a memory length of 1000*20Ch
Dim buf (1000*20) As Short
' The number of bytes of the buffer is 1000 x 20 x 2 bytes (WE_SWORD)
recSize = 1000*20*2
ret = Module1.GetAcqData (-1, 0, recSize, buf(0), ptype)
```

GetAcqDataEx

Description

Retrieves the acquisition data from the measurement module. Unlike GetAcqData, this method can read the interpolated data (Peak-to-peak (MIN-MAX) data).

Syntax

GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As SByte) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Byte) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Uint16) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Integer) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As UInt32) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Uint64) As ppNum As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Uint64) As
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Old interface

GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As SByte, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Byte, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Short, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Uint16, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Integer, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As UInt32, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Uint64, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Single, ByRef *ptype* As Short) As Short
GetAcqDataEx (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, *ppNum* As Integer, *Interpolation* As Short,ByRef *recSize* As Integer, ByRef *buf* As Double, ByRef *ptype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
blockNo (IN)	Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E).*
startPoint (IN)	Start point for the retrieval of the data. The counting origin is zero. Specifying -1 sets the start point to the top of the data.
endPoint (IN)	Last point at which to retrieve the data. The counting origin is zero. Specifying -1 sets the end point to the end of the data.
ppNum (IN)	Number of display data sets (Pair of MIN and MAX values) If -1: Data compression/interpolation is not performed. (endPoint-startPoint+1) points of data are returned. If other than -1: If ppNum > (endPoint - startPoint + 1), data interpolation takes place and ppNum points of data are created. If ppNum < (endPoint - startPoint + 1), data compression takes place and ppNum points of data are created.
Interpolation (IN)	Data interpolation type selection WE_INTER_PULSE ' Pulse interpolation WE_INTER_SIN ' Sin(x)/x interpolation WE_INTER_LINE ' Line interpolation
recSize (IN/OUT)	Data buffer size for retrieving data or the data size that was retrieved
buf (OUT)	Pointer to the data buffer
ptype (OUT)	Data type (old interface) WE_NULL ' No parameter WE_UBYTE ' 8-bit unsigned integer WE_SBYTE ' 8-bit signed integer WE_BIT8 ' 8-bit logical value WE_UWORD ' 16-bit unsigned integer WE_SWORD ' 16-bit signed integer WE_BIT16 ' 16-bit logical value WE_ULONG ' 32-bit unsigned integer WE_SLONG ' 32-bit signed integer WE_BIT32 ' 32-bit logical value WE_FLOAT ' 32-bit real number WE_DOUBLE ' 64-bit real number WE_BIT64 ' 64-bit logical value

Note:

For the free run acquisition mode, call the latch method LatchData before calling this method as necessary. The data size between latches can be obtained. For details, see *chapter 4, "Data Acquisition Model" in the WE Control API User's Manual (IM707741-61E).*

Example (Visual Basic .Net)

```
' Retrieve the acquisition data of channel 1 of the WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim recSize As Long
Dim buf(999) As Byte
Dim ptype As Integer
recSize = 1000
ret = Module1.GetAcqDataEx(1, 0, 1, 1000, -1, WE_INTER_SIN, recSize, buf(0),
ptype)
```

GetScaleData

Description

Retrieves the acquisition data of the measurement module that have been scaled. Converts the acquisition data that have been A/D converted to physical values and then scales them according to the user-specified scale parameter.

Syntax

GetScaleData (*ch* As Short, *blockNo* As Integer, ByRef *param* As ScalingParam, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetScaleData (*ch* As Short, *blockNo* As Integer, ByRef *param* As ScalingParam, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN)
Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.

blockNo (IN)
Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.

param (IN/OUT)
Scaling parameter information. If -1 is specified for ch, specify the channel worth of params. Set the params in the order of channel data to be retrieved. The scale values are used to calculate ax + b where x is the physical value of the data.

```
Structure ScalingParam
    a  As Double          ' a of the scaling parameter ax + b
    b  As Double          ' b of the scaling parameter ax + b
    ch As Integer         ' Channel number retrieved
End Structure
```

recSize (IN/OUT)
Data buffer size for retrieving data or the data size that was retrieved (bytes)

buf (OUT)
Pointer to the data buffer. Allocate enough buffer space to store the acquisition data being retrieved. Make sure to take ptype into account.

Note:

For the free run acquisition mode, call the latch method LatchData before calling this method as necessary. The data size between latches can be obtained. For details, see *chapter 4, "Data Acquisition Model" in the WE Control API User's Manual (IM707741-61E)*.

Example (Visual Basic .Net)

```
' Retrieve the physical value data of all channels of the WE7111.
' Retrieve the physical value data without scale conversion
' in 32-bit real number (4 bytes) format.
' (Specify 1 and 0 for a and b, respectively, in the scaling equation.)
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim recSize As Integer
' Allocate a buffer for a memory length of 1000 x 2ch
Dim buf (999*2) As Single
' The number of bytes of buffer is 1000 x 2ch x 4 bytes (WE_FLOAT)
recSize = 1000*2*4
Dim param (2) As ScalingParam
param (0).a = 1
param (0).b = 0
param (1).a = 1
param (1).b = 0
ret = Module1.GetScaleData (-1, 0, param(0), recSize, buf(0))
```

GetScaleDataEx

Description

Reads the data obtained after scaling the acquisition data of the measurement module. Uses the scale conversion values that were specified using the SetScaleInfo method or the ShowLinearScaleWindow method, that are stored in the module.

Syntax

GetScaleDataEx (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short

GetScaleDataEx (*ch* As Short, *blockNo* As Integer, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number One origin. If the modules are linked, it is the link number. -1 represents all channels.
blockNo (IN)	Block number For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.
recSize (IN/OUT)	Size of the received data buffer in bytes (IN)/Size of the received data in bytes (OUT)
buf (OUT)	Pointer to the data buffer to be read. Allocate enough buffer space to store the acquisition data being retrieved.

Example (Visual Basic .Net)

```
' Read the scale converted data of all the channels of the WE7111 module.
' Open the measuring station with the name "Station 1."
ret =Station.OpenStation ("Station1")
' Open the WE111 module with a link number of 2.
ret = Module1.OpenModule (Station,"WE7111:1",2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim recSize As Integer
' Allocate a buffer for a memory length of 1000 x 2ch
Dim buf (999*2) As Single
' The number of bytes of buffer is 1000 x 2ch x 4 bytes (WE_FLOAT)
recSize = 1000*2*4
ret = Module1.GetScaleDataEx (-1, 0, recSize, buf(0))
```

LatchData

Description

Issues the latch command that specifies the range of acquisition data to retrieve during the free run mode. For details, see *chapter 4, "Data Acquisition Model" in the WE Control API User's Manual (IM707741-61E)*.

Syntax

LatchData () As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

None

Example (Visual Basic .Net)

```
' Issue the latch command to WE7241.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7241 module with link number of 2.
ret = Module1.OpenModule (Station, "WE7241:1", 2)
ret = Module1.Start ()
  ......
ret = Module1.LatchData ()
```

GetCurrentData

Description

Retrieves instantaneous data from the measurement module. This method retrieves the newest data.

Syntax

GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As SByte) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Byte) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As UInt16) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Integer) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As UInt32) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Old interface

GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As SByte, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Byte, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Short, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As UInt16, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Integer, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As UInt32, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Single, ByRef *ptype* As Short) As Short
GetCurrentData (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Double, ByRef *ptype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
recSize (IN/OUT)	Data buffer size for retrieving data or the data size that was retrieved
buf (OUT)	Pointer to the data buffer. For a Description of the data types for each module, see *WeGetCurrentData in "Valid Common Measurement Control API" of each module in chapter 8, "ASCII Commands" in the WE Control API User's Manual (IM707741-61E).*
ptype (OUT)	Data type (old interface)

```
WE_NULL      ' No parameter
WE_UBYTE     ' 8-bit unsigned integer
WE_SBYTE     ' 8-bit signed integer
WE_UWORD     ' 16-bit unsigned integer
WE_SWORD     ' 16-bit signed integer
WE_ULONG     ' 32-bit unsigned integer
WE_SLONG     ' 32-bit signed integer
WE_FLOAT     ' 32-bit real number
WE_DOUBLE    ' 64-bit real number
```

Example (Visual Basic .Net)

- **Retrieve instantaneous data from all channels of the WE7251.**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1
ret = Module1.OpenModule (Station, "WE7251:1", 1)
' Waiting for completion of data acquisition after starting measurement
  ......
' Allocate Double 10 CH of data buffer.
Dim buf (10) As Double
Dim recSize As Long
Dim ptype As Integer
recSize = 10*8 ' 10CH x 8 bytes
ret = Module1.GetCurrentData (-1, recSize, buf(0), ptype)
```

- **Retrieve instantaneous data from all channels of the WE7241.**

```
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7241 module with a link number of 1
ret = Module1.OpenModule (Station, "WE7241:1", 1)
' Waiting for completion of data acquisition after starting measurement
  ......
' Allocate Single 10 CH of data buffer.
Dim buf (10) As Single
Dim recSize As Long
recSize = 10*4 ' 10CH x 4 bytes
ret = Module1.GetCurrentData (-1, recSize, buf(0))
```

Note:

Retrieval is carried out even if there is no data for one or more of the specified channels.

GetScaleCurrentData

Description

Reads the data obtained after scaling the instantaneous values of the measurement module.

Syntax

GetScaleCurrentData (*ch* As Short, ByRef *param* As ScalingParam, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetScaleCurrentData (*ch* As Short, ByRef *param* As ScalingParam, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN) Channel number. One origin. If the modules are linked, it is the link number. -1 represents all channels.

param (IN/OUT) Scale parameter information

If -1 is specified for ch, the number of Parameters that needs to be specified is equal to the number of channels. Set the params in the order of channel data to be retrieved. The scale values are used to calculate ax + b where x is the physical value of the data.

```
Structure ScalingParam
    a As Double          ' a of the scaling parameter ax + b
    b As Double          ' b of the scaling parameter ax + b
    ch As Long           ' Channel number retrieved
End Structure
```

recSize (IN/OUT) Size of the received data buffer in bytes (IN)/Size of the received data in bytes (OUT)

buf (OUT) Pointer to the data buffer to be read.

Example (Visual Basic .Net)

```
Read the data obtained after scaling the instantaneous value of the WE7271
module.
' Open the measuring station with the name "Station 1."
ret =Station.OpenStation ("Station1")
' Open the WE7271 module with a link number of 1.
ret = Module1.OpenModule (Station,"WE7271:1",1)
' Waiting for completion of data acquisition after starting measurement
  ......
' Allocate a buffer for a memory length of Double 4ch
Dim buf (3) As Double
Dim recSize As Long
recSize = 4*8          ' 4ch x 8 bytes
Dim param(3) As ScalingParam
param(0).a =1.0
param(0).b =0.0
param(1).a =2.0
param(1).b =0.0
param(2).a =3.0
param(2).b =0.0
param(3).a =4.0
param(3).b =0.0
ret = Module1.GetScaleCurrentData (-1, param(0), recSize, buf(0))
```

Note:

Channels not specified for measurement are also included in the data, so you must specify enough buffers to accommodate those channles as well.

GetScaleCurrentDataEx

Description

Reads the data obtained after scaling the instantaneous values of the measurement module. Uses the scale conversion values that were specified using the SetScaleInfo method or the ShowLinearScaleWindow method, that are stored in the module.

Syntax

GetScaleCurrentDataEx (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Single) As Short
GetScaleCurrentDataEx (*ch* As Short, ByRef *recSize* As Integer, ByRef *buf* As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number One origin. If the modules are linked, it is the link number. -1 represents all channels.
recSize (IN/OUT)	Size of the received data buffer in bytes (IN)/Size of the received data in bytes (OUT)
buf (OUT)	Pointer to the data buffer to be read.

Example (Visual Basic .Net)

```
Read the data obtained after scaling the instantaneous value of the WE7271
module.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7271 module with a link number of 1.
ret = Module1.OpenModule (Station,"WE7271:1",1)
' Waiting for completion of data acquisition after starting measurement
  ......
' Allocate a buffer for a memory length of Double 4ch
Dim buf (3) As Double
Dim recSize As Long
recSize = 4*8          ' 4ch x 8 bytes
ret = Module1.GetScaleCurrentDataEx (-1, recSize, buf(0))
```

Note:

Channels not specified for measurement are also included in the data, so you must specify enough buffers to accommodate those channles as well.

GetMeasureParam

Description

Gets the automated measurement values from the measurement module. The automated measurement values are the results analyzed by the modules. For modules that do not have the automated measurement method, the ExecMeasureParam method can be used.

Syntax

GetMeasureParam (*ch* As Short, *blockNo* As Integer, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem, ByRef *itemNum* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN) Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.

blockNo (IN) Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.

startPoint (IN) Start point of the data for the automated measurement. One origin. Specifying -1 sets the start point to the top of the data.

endPoint (IN) End point of the data for the automated measurement. One origin. Specifying -1 sets the end point to the end of the data.

item (OUT) Measurement information pointer

```
Structure MeasureItem     ' Measurement result storage structure
    Channel As Double     ' CH number (1 or greater)
    Max As Double         ' Maximum value
    Min As Double         ' Minimum value
    High As Double        ' High level
    Low As Double         ' Low level
    PP As Double          ' P-P value
    Ampl As Double        ' Amplitude
    Avg As Double         ' Average value
    Rms As Double         ' RMS value
    Middle As Double      ' Center value of the amplitude
    StdDev As Double      ' Standard deviation
    Oshoot As Double      ' Overshoot
    Ushoot As Double      ' Undershoot
    Rise As Double        ' Rise time
    Fall As Double        ' Fall time
    Freq As Double        ' Frequency
    Period As Double      ' Period
    Duty1 As Double       ' Duty cycle on the High side
    Duty2 As Double       ' Duty cycle on the Low side
    Width1 As Double      ' Width above the mesial value
    Width2 As Double      ' Width below the mesial value
End Structure
```

itemNum (OUT) Number of MeasureItem that are actually retrieved

Example (Visual Basic .Net)

```
' Get the measurement data of channel 1 of the WE7111.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim buf As MeasureItem
Dim num As Integer
ret = Module1.GetMeasureParam (1, 0, 1, 10000, buf, num)
```

SaveAcqData

Description

Saves the acquisition data of the measurement module to a file.

Syntax

SaveAcqData (*ch* As Short, *blockNo* As Integer, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin is. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
blockNo (IN)	Block number. If the number exceeds the valid number of blocks, all block data are saved. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.
filename (IN)	File name. The file extension is not necessary. A file with the extension, WVF, is created.
htype (IN)	Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1: create).

Example (Visual Basic .Net)

```
' Save the raw data of channel 1 of the WE7111 to a file.
' Create a header file.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
ret = Module1.SaveAcqData (1, 0, "c:\dsoparam",1)
```

SaveScaleData

Description

Saves the scaled acquisition data of the measurement module to a file.

Syntax

SaveScaleData (*ch* As Short, *blockNo* As Integer, ByRef *param* As ScalingParam, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
blockNo (IN)	Block number. For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.
param (IN/OUT)	Scaling parameter information Structure ScalingParam a As Double ' a of the scaling parameter ax + b b As Double ' b of the scaling parameter ax + b ch As Long ' Channel number retrieved End Structure
filename (IN)	File name. The file extension is not necessary. A file with the extension, WVF, is created.
htype (IN)	Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1: create).

Example (Visual Basic .Net)

```
' Save the physical value data of channel 1 of the WE7111.
' Create a header file.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
Dim param (1) As ScalingParam
param (0).a = 1.0
param (0).b = 0.0
param (1).a = 1.0
param (1).b = 0.0
ret = Module1.SaveScaleData (1, 0, param(0), "c:\dsoparam", 1)
```

SaveScaleDataEx

Description

Saves the scaled acquisition data of the measurement module to a file in binary format. Uses the scale conversion values that were specified using the SetScaleInfo method or the ShowLinearScaleWindow method, that are stored in the module.

Syntax

SaveScaleDataEx (*ch* As Short, *blockNo* As Integer, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number One origin. If the modules are linked, it is the link number. -1 represents all channels.
blockNo (IN)	Block number For details, see *section 4.2, "Specifying the Data Block that You Wish to Retrieve" in the WE Control API User's Manual (IM 707741-61E)*.
filename (IN)	File name. The file extension is not necessary. A file with the extension, WVF, is created.
htype (IN)	Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1: create).

Example (Visual Basic .Net)

```
' Save the raw data of all the channels of the WE7111 to a file.
' Create a header file.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE111 module with a link number of 2.
ret = Module1.OpenModule (Station,"WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
ret = Module1.SaveScaleDataEx (-1, 0, "c:\ WE7111Data", 1)
```

SaveAsciiData

Description

Saves the physical value data of the measurement module to a file in ASCII format (CSV format). the physical value data is the measurement unit data of the module. This is not the scaled acquisition data. The measurement unit of the digitizer module is "voltage," for example.

Syntax

SaveAsciiData (*ch* As Short, *blockNo* As Integer, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
blockNo (IN)	Block number
filename (IN)	File name. The file extension is not necessary. A file with the extension, CSV, is created.
htype (IN)	Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1:create).

Example (Visual Basic .Net)

```
' Save the physical value data of channel 1 of the WE7111 in
' ASCII format.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
ret = Module1.SaveAsciiData (1, 0, "c:\dsodata", 1)
```

SaveScaleAsciiData

Description

Saves the scaled acquisition data of the measurement module to a file in ASCII format (CSV format).

Syntax

SaveScaleAsciiData (*ch* As Short, *blockNo* As Integer, ByRef *param* As ScalingParam, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN) Channel number
One origin. If the modules are linked, it is the link number. -1 represents all channels.

blockNo (IN) Block number

param (IN/OUT) Scale value information
If -1 is specified for ch, the number of Parameters that needs to be specified is equal to the number of channels. Set the params in the order of channel data to be retrieved. The scale values are used to calculate ax + b where x is the physical value of the data.

```
Structure ScalingParam
    a  As Double          ' a of the scaling parameter ax + b
    b  As Double          ' b of the scaling parameter ax + b
    ch As Long            ' Channel number retrieved
End Structure
```

filename (IN) File name.
The file extension is not necessary. A file with the extension, CSV, is created.

htype (IN) Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1: create).

Example (Visual Basic .Net)

```
' Save the scaled physical value data of all the channels of the WE7111 in
' ASCII format.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE111 module with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
Dim param(1)As ScalingParam
param(0).a =1.0
param(0).b =0.0
param(1).a =2.0
param(1).b =0.0
' Waiting for completion of data acquisition after starting measurement
  ......
ret = Module1.SaveScaleAsciiData (-1, 0, param(0), "c:\WE7111Data", 1)
```

SaveScaleAsciiDataEx

Description

Saves the scaled acquisition data of the measurement module to a file in ASCII format (CSV format). Uses the scale conversion values that were specified using the SetScaleInfo method or the ShowLinearScaleWindow method, that are stored in the module.

Syntax

SaveScaleAsciiDataEx (*ch* As Short, *blockNo* As Integer, *filename* As String, *htype* As Short) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number One origin. If the modules are linked, it is the link number. -1 represents all channels.
blockNo (IN)	Block number
filename (IN)	File name. The file extension is not necessary. A file with the extension, CSV, is created.
htype (IN)	Whether or not to create the waveform information file (HDR file extension) for the acquisition data (0: do not create, 1: create).

Example (Visual Basic .Net)

```
' Save the scaled physical value data of all the channels of the WE7111 in
' ASCII format.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE111 module with a link number of 2.
ret = Module1.OpenModule (Station,"WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
ret = Module1.SaveScaleAsciiDataEx (-1, 0, "c:\WE7111Data", 1)
```

SaveAcqHeader

Description

Creates the waveform information file for the acquisition data of the measurement module. The file that is created is the same file created with the SaveAcqData method.

Syntax

SaveAcqHeader (*ch* As Short, *filename* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful. If the number exceeds the valid number of blocks, an error occurs.

Parameters

ch (IN)	Channel number to save. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
filename (IN)	File name. The file extension is not necessary. A file with the extension, HDR, is created.

Example (Visual Basic .Net)

```
' Save the waveform information file for all channels of the WE7111
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open module WE7111 with a link number of 2.
ret = Module1.OpenModule (Station, "WE7111:1", 2)
' Waiting for completion of data acquisition after starting measurement
  ......
ret = Module1.SaveAcqHeader (-1, "c:\station")
```


SavePatternData

Description

Saves the pattern data that are dependent on the measurement module to a file. The pattern data are different for each module. Some modules do not have pattern data defined. Pattern data are, for example, the arbitrary waveform data for the WE7121 and the digital pattern data for the WE7131.

Syntax

SavePatternData (*ch* As Short, *filename* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
filename (IN)	File name. The file extension is not necessary. The file extension defined for each measurement module is automatically added.

Example (Visual Basic .Net)

```
' Save the pattern data of channel 1 of the WE7131.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7131 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7131:1",1)
ret = Module1.SavePatternData (1, "c:\piopattern")
```

LoadPatternData

Description

Loads the pattern data that are dependent on the measurement module . The pattern data are different for each module. Some modules do not have pattern data defined. Pattern data are, for example, the arbitrary waveform data for the WE7121 and the digital pattern data for the WE7131.

Syntax

LoadPatternData (*ch* As Short, *filename* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

ch (IN)	Channel number. One origin. If the modules are linked, the channel number spans across the linked modules. -1 represents all channels.
filename (IN)	File name

Example (Visual Basic .Net)

```
' Load the arbitrary waveform data to channel 1 of the WE7121.
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7121 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7121:1",1)
ret = Module1.LoadPatternData (1, "c:\fgarb")
```

LoadPatternDataEx

Description

Loads the waveform data that are retrieved using the specified parameter from the specified file (wvf or csv format) to the measurement module.

Syntax

LoadPatternDataEx (*command* As String, *filename* As String, *ch* As Short, *blockNo* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

command (IN)	ASCII command
filename (IN)	File name
ch (IN)	Channel number. One origin.
blockNo (IN)	Block number. Zero origin.

Note:

By setting the channel number to 0x7FFF or the block number to 0x7FFFFFFF, the waveform data of all channels or all blocks are transferred.
For the WE7281 module, auto load (AG:Misc:Load:Auto command) must be turned On.

Example (Visual Basic .Net)

• Example in which the LoadPatternDataEx method is used

```
' Transfer the Block0 data of CH1 of the wvf file saved by the
' WE7271 module to CH1 of the WE7281 module.
ret = module1.LoadPatternDataEx ("FG:CH1:Load ARB", "c:\we7271.wvf", 1, 0)
```

• Example in which the Filter method is used

```
Dim size As Integer
' Determine the byte size when the Block0 data of CH1 of the wvf
' file saved by the WE7271 module are converted to the format used
' by the FG mode on the WE7281 module.
ret = Filter.Wvf2S16GetSize("c:\we7271.wvf", 1, 0, size)
' Allocate the buffer using the retrieved data size.
size = size/2
ReDim s16buf(size) As Short
' Convert the Block0 data of CH1 of the wvf file saved by the
' WE7271 module to the format used by the FG mode on the WE7281 module.
ret = Filter.Wvf2S16("c:\we7271.wvf", 1, 0, s16buf(0), size)
' Transfer the converted data to CH1 of the WE7281 module.
ret = Module1.SetControl ("FG:CH1:Load ARB", s16buf)
```

SetOverRun

Description

Select whether or not to detect overruns.

Syntax

SetOverRun (*sw* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

sw (IN) Overrun detection setting
0 ' Do not detect overruns (do not stop when an overrun occurs)
1 ' Detect overruns (stop when an overrun occurs)

Note:

Overruns are detected as default.

Example (Visual Basic .Net)

```
' Disable the overrun detection on the WE7241 module.
' Open the station handle.
ret = Station.OpenStation ("Station1")
' Open module WE7241.
ret = Module1.OpenModule (Station, "WE7241:1", 1)
' Disable the overrun detection on the WE7241 module.
ret = Module1.SetOverRun (hMo, 0)
```

GetOverRun

Description

Queries whether or not overruns are to be detected.

Syntax

GetOverRun (ByRef *sw* As Byte) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

sw (IN) Overrun detection setting
0 ' Do not detect overruns (do not stop when an overrun occurs)
1 ' Detect overruns (stop when an overrun occurs)

Example (Visual Basic .Net)

```
' Queries the overrun detection setting on the WE7241 module.
Dim sw As Byte
' Open the station handle.
ret = Station.OpenStation ("Station1")
' Open module WE7241.
ret = Module1.OpenModule (Station, "WE7241:1", 1)
' Queries the overrun detection setting.
ret = Module1.GetOverRun (sw)
```

CreateEvent

Description

Creates a handle for handling measurement module events and enables the operation. Up to 32 handles can be created for each measurement module. The cause of events are module common events and module dependent events. For a Description of the module-dependent events, see the command table of each module in the WE Control API User's Manual (IM707741-61E).

Syntax

CreateEvent (*pattern* As Integer, *mode* As Integer, ByRef *evHandle* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

pattern (IN) Causes of events. The following common events are defined. Multiple events can be specified. There are other module dependent events that are defined beside the one listed below. For details, see *chapter 8, "ASCII Commands" in the WE Control API User's Manual (IM707741-61E)*.

WE_EV_MEASSTART	' Generated when the start operation ' completes
WE_EV_MEASEND	' Generated when the specified number of ' blocks of data are acquired.
WE_EV_MEASABORT	' Generated when the start state (RUN state) ' is cleared.
WE_EV_BLOCKEND	' Generated each time data are acquired to ' the block when acquiring data using ' memory partitions (blocks).
WE_EV_ERROR	' Generated when an error defined by the ' measurement module occurs. ' For details regarding the errors see the ' command table for each module.

mode (IN)	Event operation mode	
	WE_EV_STOP	' Disable the event operation.
	WE_EV_CONT	' Detect and generate the event repetitively.
	WE_EV_SINGLE	' Detect and generate the event once.
	WE_EV_SINGLE_RELEASE	' Detect and generate the event once ' and then release the event handle.
evHandle (OUT)	Event handle retrieved	

Note:

Handles do not need to be created for the power event (WE_EV_POWER) and the fan stop event (WE_EV_FANSTOP). These events are always detected.

Example (Visual Basic .Net)

```
' Set (Enable) the block end event for the WE7251.
Dim ret As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7251", 1)
' Set the block end event.
ret = Module1.CreateEvent (WeControl.WE_EV_BLOCKEND, WeControl.WE_EV_CONT,
evHandle)
```

SetEventPattern

Description

Set the cause of an event of the measurement module.

Syntax

SetEventPattern (*evHandle* As Integer, *pattern* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

evHandle (IN)	Event handle (0 to 31)
pattern (IN)	Cause of an event

Example (Visual Basic .Net)

```
' Set (Enable) the block end event for the WE7251.
Dim evHandle As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7251", 1)
' Set the measurement end event.
ret = Module1.CreateEvent(0, WeControl.WE_EV_CONT, evHandle)
ret = Module1.SetEventPattern(evHandle, WeControl.WE_EV_BLOCKEND)
```

ResetEventPattern

Description

Resets the cause of an event of the measurement module.

Syntax

ResetEventPattern (*evHandle* As Integer, *pattern* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

evHandle (IN)	Event handle (0 to 31)
pattern (IN)	Cause of an event

Example (Visual Basic .Net)

```
' Reset (Clear) the block end event for the WE7251.
Dim evHandle As Integer
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7251", 1)
ret = Module1.CreateEvent (WeControl.WE_EV_BLOCKEND, WeControl.WE_EV_CONT,
evHandle)
ret = Module1.ResetEventPattern (evHandle, WeControl.WE_EV_BLOCKEND)
```

SetEventMode

Description

Set the operation mode of the measurement module events.

Syntax

SetEventMode (*evHandle* As Integer, *mode* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

evHandle (IN)	Event handle (0 to 31)	
mode (IN)	Event operation mode	
	WE_EV_STOP	' Disable the event operation.
	WE_EV_CONT	' Detect and generate the event repetitively.
	WE_EV_SINGLE	' Detect and generate the event once.
	WE_EV_SINGLE_RELEASE	' Detect and generate the event once and ' then release the event handle.

Example (Visual Basic .Net)

```
' Disable the event operation mode for the WE7251.
Dim evHandle As Long
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7251", 1)
ret = Module1.CreateEvent (WeControl.WE_EV_BLOCKEND, WeControl.WE_EV_CONT,
evHandle)
ret = Module1.SetEventMode (evHandle, WeControl.WE_EV_STOP)
```


ReleaseEvent

Description

Releases the event handle of the measurement module. The handle cannot be used after it is released. However, the number can be used to create a new handle.

Syntax

ReleaseEvent (*evHandle* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

evHandle (IN) Event handle (0 to 31)

Example (Visual Basic .Net)

```
' Set the block end event for the WE7251 and then release the handle.
Dim evHandle As Long
' Open the measuring station with the name "Station 1."
ret = Station.OpenStation ("Station1")
' Open the WE7251 module with a link number of 1.
ret = Module1.OpenModule (Station, "WE7251", 1)
' Set the block end event.
ret = Module1.CreateEvent(WeControl.WE_EV_BLOCKEND, WeControl.WE_EV_CONT,
evHandle)
ret = Module1.ReleaseEvent (evHandle)
```

3.5 WeLib Class

ExecMeasureParam

Description

Performs automated computations on physical value data. GetMeasureParam method performs the automated computation of waveform Parameters in the measurement module, but this method computes the parameters on the PC.

Syntax

ExecMeasureParam (ByRef *data* As Single, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParam (ByRef *data* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

data (IN)	The computed data of waveform parameter. The data is physical value.
dt (IN)	Sampling interval (seconds)
startPoint (IN)	Starting point of the data to compute the parameter values. One origin.
endPoint (IN)	Last point of the data to compute the parameter values. One origin. Calculates the parameter values using the data in the range from startPoint to endPoint.
item (OUT)	Waveform parameter pointer

```
Structure MeasureItem      ' Computation result storage structure
   Channel As Double       ' CH number (0)
   Max As Double           ' Maximum value
   Min As Double           ' Minimum value
   High As Double          ' High level
   Low As Double           ' Low level
   PP As Double            ' P-P value
   Ampl As Double          ' Amplitude
   Avg As Double           ' Average value
   Rms As Double           ' RMS value
   Middle As Double        ' Center value of the amplitude
   StdDev As Double        ' Standard deviation
   Oshoot As Double        ' Overshoot
   Ushoot As Double        ' Undershoot
   Rise As Double          ' Rise time
   Fall As Double          ' Fall time
   Freq As Double          ' Frequency
   Period As Double        ' Period
   Duty1 As Double         ' Duty cycle on the High side
   Duty2 As Double         ' Duty cycle on the Low side
   Width1 As Double        ' Width above the mesial value
   Width2 As Double        ' Width below the mesial value
End Structure
```

Example (Visual Basic .Net)

```
' Retrieves the measurement results of 1000 points of data
' at a sampling rate of 0.5 sec.
Dim item As Measureitem
Dim data (1000) As Single
ret = Lib1.ExecMeasureParam (data(0), 0.5, 1, 1000, item)
```

ExecMeasureParamAcqData

Description

Performs automated computations on the acquisition data (data after A/D conversion). GetMeasureParam method performs the automated computation of waveform Parameters in the measurement module, but this method computes the parameters on the PC.

Syntax

ExecMeasureParamAcqData (ByRef *data* As SByte, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParamAcqData (ByRef *data* As Byte, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParamAcqData (ByRef *data* As Short, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParamAcqData (ByRef *data* As UInt16, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParamAcqData (ByRef *data* As Integer, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short
ExecMeasureParamAcqData (ByRef *data* As UInt32, *gain* As Double, *ofst* As Double, *dt* As Double, *startPoint* As Integer, *endPoint* As Integer, ByRef *item* As MeasureItem) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

data (IN)	Pointer to the data on which to perform the automated computation
gain (IN)	Gain (Range)
ofst (IN)	Offset Converts the data to physical values according to the following equation. X is the value of the acquisition data. gain x X + ofst
dt (IN)	Sampling interval (seconds)
startPoint (IN)	Starting point of the data to be used to compute the parameter values.
endPoint (IN)	Last point of the data to be used to compute the parameter values. Calculates the parameter values using the data in the range from startPoint to endPoint.

item (OUT) Waveform parameter pointer

```
Structure MeasureItem     ' Computation result storage structure
   Channel As Double      ' CH number
   Max As Double          ' Maximum value
   Min As Double          ' Minimum value
   High As Double         ' High level
   Low As Double          ' Low level
   PP As Double           ' P-P value
   Ampl As Double         ' Amplitude
   Avg As Double          ' Average value
   Rms As Double          ' RMS value
   Middle As Double       ' Center value of the amplitude
   StdDev As Double       ' Standard deviation
   Oshoot As Double       ' Overshoot
   Ushoot As Double       ' Undershoot
   Rise As Double         ' Rise time
   Fall As Double         ' Fall time
   Freq As Double         ' Frequency
   Period As Double       ' Period
   Duty1 As Double        ' Duty cycle on the High side
   Duty2 As Double        ' Duty cycle on the Low side
   Width1 As Double       ' Width above the mesial value
   Width2 As Double       ' Width below the mesial value
End Structure
```

Example (Visual Basic .Net)

```
' Retrieve the Computation result of 1000 points of data at
' sampling rate of 0.5 sec, gain of 1.0, and offset of 0.0.
Dim item As Measureitem
Dim data (1000) As Single
ret = Lib1.ExecMeasureParam AcqData(data(0), 1.0, 0.0, 0.5, 1, 1000, item)
```

GetHandle

Description

Gets the module handle from the second parameter (station number in the upper 16 bits and the module number in the lower 16 bits) of the event explained in WeModule Class.

Syntax

GetHandle (*param* As Integer, ByRef *hMo* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

param (IN) The value of the second parameter of the event (lparam).
hMo (OUT) Module handle to be retrieved.

Example (Visual Basic .Net)

```
' Get the module handle.
Dim hMo As Long
ret = Lib1.GetHandle (param, hMo)
```

IsNan

Description

Queries whether or not the data are non-numeric.

Syntax

IsNan (*value* As Single) As Short
IsNan (*value* As Double) As Short

Return value

Returns 0 when it is not non-numeric. Returns a nonzero number otherwise.

Parameters

value (IN)	Data specified

Example (Visual Basic .Net)

```
' Check whether or not a value is non-numeric.
Dim data As Single
data = 9999999
ret = Lib1.IsNan (data)
```

GetAlarmInfo

Description

Gets the alarm information.

Syntax

GetAlarmInfo(*dp* As Integer, ByRef *buf* As Integer, *size* As Integer, *handle* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

dp (IN)	Data to be specified. Specifies member dp of the second parameter e of the WeAlarm event handler.
buf (OUT)	Alarm information storage buffer
size (IN)	Data size
handle (IN)	Specifies member handle of the second parameter e of the WeAlarm event handler.

Example (Visual Basic .Net)

```
Private Sub AxWeEvent1_WeAlarm(ByVal sender As Object,
ByVal e As AxWEEVENTLib._DWeEventEvents_WeAlarmEvent)
Handles AxWeEvent1.WeAlarm
    Dim hMo As Integer
    Dim recSize As Integer
    Dim i As Integer
    Dim buf() As Integer

    ret = Lib1.GetHandle(e.handle, hMo)    ' Get the handle
    If hMo = Module1.hMo Then
      Lib1.MoveMemory(recSize, e.dp, 4)    ' Get the data size
        If recSize <> 0 Then
            ReDim buf(recSize)             ' Reserve area
            ret = Lib1.GetAlarmInfo(e.dp, buf(0), recSize, e.handle)
            ' Get alarm information
            For i = 0 To recSize - 1
           ' Display the alarm determination result and alarm determination
            channel information on the debug window.
               Console.WriteLine("Alarm information=" + Hex(buf(i)))
            Next
        End If
    End If
End Sub
```

MoveMemory

Description

Copies the memory contents.

Syntax

MoveMemory(ByRef *dstdt* As Integer, *srcdt* As Integer, *length* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

dstdt (OUT)	Pointer to the copy destination buffer.
srcdt (IN)	Pointer to the source data buffer. (Passes the address by value.)
length (IN)	Data size

Example (Visual Basic .Net)

```
Private Sub AxWeEvent1_WeAlarm(ByVal sender As Object,
ByVal e As AxWEEVENTLib._DWeEventEvents_WeAlarmEvent)
Handles AxWeEvent1.WeAlarm
    Dim hMo As Integer
    Dim recSize As Integer
    Dim i As Integer
    Dim buf() As Integer

    ret = Lib1.GetHandle(e.handle, hMo)    ' Get the handle
    If hMo = Module1.hMo Then
      Lib1.MoveMemory(recSize, e.dp, 4)    ' Get the data size
        If recSize <> 0 Then
            ReDim buf(recSize)             ' Reserve area
            ret = Lib1.GetAlarmInfo(e.dp, buf(0), recSize, e.handle)
            ' Get alarm information
            For i = 0 To recSize - 1
           ' Display the alarm determination result and alarm determination
             channel information on the debug window.
               Console.WriteLine("Alarm information=" + Hex(buf(i)))
            Next
        End If
    End If
End Sub
```

TransAcqData

Description

The GetAcqData method converts the acquisition data (raw data) retrieved using the GetAcqDataEx method according to the data information. The data information can be retrieved using the GetAcqDataInfo method. The raw data of some modules must be converted using this method. For a Description of the modules that requires conversion, see the note.

Syntax

TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As SByte, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Byte, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Short, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As UInt16, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Integer, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As UInt32, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Single, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Double, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As Long, ByRef *info* As AcqDataInfo) As Short
TransAcqData(*chNum* As Short, *recsize* As Integer, ByRef *buf* As UInt64, ByRef *info* As AcqDataInfo) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

chNum (IN)	Number of channels (1 or greater)
recsize (IN)	Data size (bytes) of the raw data received using GetAcqData/GetAcqDataEx
buf (IN/OUT)	Stores the pointer to the raw data buffer (IN)/data after the conversion (OUT)
info (IN/OUT)	Stores the pointer to the data information retrieved using GetAcqDataInfo (IN)/ the data information after conversion (OUT)

Note:

This method performs bit extraction and inversion on the acquisition data according to the StartBit and EffectiveBit values in the data information, and then sets the type of data acquisition data after the conversion (dataType) in the data information. The data size of the acquisition data type does not change. (For example, change from WE_ULONG to WE_FLOAT is possible, but WE_ULONG to WE_DOUBLE is not.)

The data that needs to be converted using this method is raw data retrieved using the GetAcqData method or GetAcqDataEx method. The SaveAcqData method saves data that has been converted in advance. Thus, the data does not need to be converted using this method.

Below are the modules that require the raw data to be converted using this method.

- WE7081: Performs bit extraction.
- WE7521: Performs inversion for frequency measurement data.

Example (Visual Basic .Net)

- **Read and convert the data of all channels (when 4 channels are measured) on the WE7081**

```
' Open the measuring station with the name "Station1."
ret = Station.OpenStation("Station1")
' Open the WE7081 module
ret = Module1.OpenModule(Station, "WE7081:1",1)
' Waiting for completion of data acquisition after starting measurement
Dim recSize As Integer
Dim info(3) As AcqDataInfo
Dim num As Short
' Allocate a buffer for a memory length of 1000 x 4ch
Dim buf(3999) As Double
' The number of bytes of buffer is 1000 x 4 x 8 bytes (WE_DOUBLE)
recSize = 1000*4*8
' Retrieve data information and raw data and convert the data
ret = Module1.GetAcqDataInfo(–1, 0, info(0), num)
ret = Module1.GetAcqData(–1, 0, recSize, buf(0))
ret = Lib1.TransAcqData(num, recSize, buf(0), info(0))
```

- **Read and convert the data of CH1 on the WE7521**

```
' Open the measuring station with the name "Station1."
ret = Station.OpenStation("Station1")
' Open the WE7521 module
ret = Module1.OpenModule(Station, "WE7521:1",1)
' Waiting for completion of data acquisition after starting measurement
Dim recSize As Integer
Dim info As AcqDataInfo
Dim num As Short
' Allocate a buffer for a memory length of 1000 points
Dim buf(999)As Single
' The number of bytes of buffer is 1000 x 4 bytes (WE_FLOAT)
recSize = 1000*4
' Retrieve data information and raw data and convert the data
ret = Module1.GetAcqDataInfo( 1, 0, info, num)
ret = Module1.GetAcqData(1, 0, recSize, buf(0))
ret = Lib1.TransAcqData(1, recSize, buf(0), info)
```

3.6 WeFilter Class

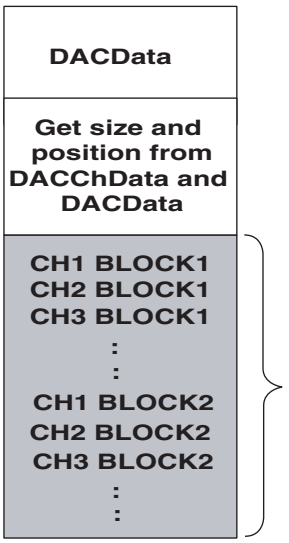
Description

This section describes the interface functions used to convert waveform data in wvf or csv format to the arbitrary waveform or sweep waveform data formats for the 4-CH, 100 kS/s D/A Module WE7281. These functions are used along with the WE Control API when transferring waveform data to the WE7281 module.

Data format

The WE7281 uses data in the following formats.

Data format	Description
s16	Arbitrary waveform data for the FG mode short x 65536
w32	Sweep waveform data for the FG mode UINT x 65536
w7281	Arbitrary waveform data for the AG mode (See figure below.)




```
typedef struct
{
        float    range ;
        intData  Length;
} DACChData;
typedef struct
{
            int  Magic;      //Substitute "DACS"=0x44414353
unsigned byte    chnum;      //Number of channel
unsigned byte    blockNum;   //Number of block
unsigned byte    pad[2];     //Padding
        double   Sampling;   //Sampling interval
} DACData ;
```

Wvf2S16GetSize

Description

Gets the byte size when waveform data retrieved using the specified parameter from the specified file (wvf or csv format), are converted to the arbitrary waveform data format for the FG mode.

Syntax

Wvf2S16GetSize(*filename* As String, *ch* As Short, *block* As Integer, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one.
block (IN)	Block number of the input file. The counting origin is zero.
size (OUT)	Data size (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' Get the byte size when the Block0 data of CH1 of the wvf file
' saved by the WE7271 module are converted to the format used by
' the FG mode on the WE7281 module.
ret = Filter.Wvf2S16GetSize("c:\we7271.wvf", 1, 0, size)
```

Wvf2W32GetSize

Description

Gets the byte size when waveform data retrieved using the specified parameter from the specified file (wvf or csv format) are converted to the sweep waveform data format for the FG mode.

Syntax

Wvf2W32GetSize(*filename* As String, *ch* As Short, *block* As Integer, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one.
block (IN)	Block number of the input file. The counting origin is zero.
size (OUT)	Data size (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' Get the byte size when the Block0 data of CH1 of the wvf file
' saved by the WE7271 module are converted to the sweep waveform
' data format used by the FG mode on the WE7281 module.
ret = Filter.Wvf2W32GetSize("c:\we7271.wvf", 1, 0, size)
```

Wvf2W7281GetSize

Description

Gets the byte size when waveform data retrieved using the specified parameter from the specified file (wvf or csv format), are converted to the arbitrary waveform data format for the AG mode.

Syntax

Wvf2W7281GetSize(*filename* As String, *ch* As Short, *block* As Integer, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one. Note that "&H7FFF" represents all channels.
block (IN)	Block number of the input file. The counting origin is zero. Note that "&H7FFFFFFF" represents all blocks.
size (OUT)	Data size (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' Get the byte size when the Block0 data of CH1 of the wvf file
' saved by the WE7271 module are converted to the format used by
' the AG mode on the WE7281 module.
ret = Filter.Wvf2W7281GetSize("c:\we7271.wvf", 1, 0, size)
```

Wvf2S16

Description

Converts waveform data retrieved using the specified parameter from the specified file (wvf or csv format), to the arbitrary waveform data format for the FG mode.

Syntax

Wvf2S16(*filename* As String, *ch* As Short, *block* As Integer, ByRef *buf* As Short, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one.
block (IN)	Block number of the input file. The counting origin is zero.
buf (OUT)	Pointer to the data buffer.
size (IN/OUT)	Data buffer size/Actual size of the data retrieved (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' The byte size when the Block0 data of CH1 of the wvf file saved
' by the WE7271 module are converted to the s16 format used by the
' FG mode on the WE7281 module.
ret = Filter.Wvf2S16GetSize("c:\we7271.wvf", 1, 0, size)
' Allocate the buffer using the retrieved data size.
size = size/2
ReDim s16buf(size) As Short
' Convert the Block0 data of CH1 of the wvf file saved by the
' WE7271 module to the format used by the FG mode on the WE7281
' module.
ret = Filter.Wvf2S16("c:\we7271.wvf", 1, 0, s16buf(0), size)
' Transfer the converted data to CH1 of the WE7281 module.
ret = Module1.SetControl ("FG:CH1:Load ARB", s16buf)
```

Wvf2W32

Description

Converts waveform data retrieved using the specified parameter from the specified file (wvf or csv format), to the sweep waveform data format for the FG mode.

Syntax

Wvf2W32(*filename* As String, *ch* As Short, *block* As Integer, ByRef *buf* As Integer, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one.
block (IN)	Block number of the input file. The counting origin is zero.
buf (OUT)	Pointer to the data buffer.
size (IN/OUT)	Data buffer size/Actual size of the data retrieved (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' The byte size when the Block0 data of CH1 of the wvf file saved
' by the WE7271 module are converted to the w32 format used by the
' FG mode on the WE7281 module.
ret = Filter.Wvf2W32GetSize("c:\we7271.wvf", 1, 0, size)
' Allocate the buffer using the retrieved data size.
size = size/4
ReDim w32buf(size) As Integer
' Convert the Block0 data of CH1 of the wvf file saved by the
' WE7271 module to the format used by the FG mode on the WE7281
' module.
ret = Filter.WeWvf2W32("c:\we7271.wvf", 1, 0, w32buf(0), size)
' Transfer the converted data to CH1 of the WE7281 module.
ret = Module1.WeSetControl ("FG:CH1:Load ARB", w32buf)
```

Wvf2W7281

Description

Converts waveform data retrieved using the specified parameter from the specified file (wvf or csv format), to the arbitrary waveform data format for the AG mode.

Syntax

Wvf2W7281(*filename* As String, *ch* As Short, *block* As Integer, ByRef *buf* As Byte, ByRef *size* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

filename (IN)	Name of the file containing the waveform data (wvf or csv format).
ch (IN)	Channel number of the input file. The counting origin is one. Note that "&H7FFF" represents all channels.
block (IN)	Block number of the input file. The counting origin is zero. Note that "&H7FFFFFFF" represents all channels.
buf (OUT)	Pointer to the data buffer.
size (IN/OUT)	Data buffer size/Actual size of the data retrieved (byte size).

Example (Visual Basic .Net)

```
Dim size As Integer
' The byte size when the Block0 data of CH1 of the wvf file saved
' by the WE7271 module are converted to the format used by the
' AG mode on the WE7281 module.
ret = Filter.Wvf2W7281GetSize("c:\we7271.wvf", 1, 0, size)
' Allocate the buffer using the retrieved data size.
ReDim w7281buf(size) As Byte
' Convert the Block0 data of CH1 of the wvf file saved by the
' WE7271 module to the format used by the AG mode on the WE7281
' module.
ret = Filter.Wvf2W7281("c:\we7271.wvf", 1, 0, w7281buf(0), size)
' Transfer the converted data to CH1 of the WE7281 module.
ret = Module1.SetControl ("AG:Misc:Load Data", w7281buf)
```

3.7 WeFile Class

HeaderReadS

Description

Reads the data from the header file by specifying the block number.

Syntax

HeaderReadS(*FileName* As String, *BlockNo* As Integer, ByRef *ComBuff* As CommonInf1, *ChBuff()* As ChanelInf1) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Header file name.
BlockNo (IN)	Block number. Specify the block number you wish to read.
ComBuff() (OUT)	Common information structure. Define using an array.
ChBuff() (OUT)	Channel information structure. Define using an array.

Note:

Collectively retrieves the header file information for data of which the number of X-axis data points is the same. Prepare a single-element array for the common information structure buffer. Prepare channel information structure buffer for the amount equal to the number of channels.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNo As Integer
Dim ComBuff As CommonInf1
Dim ChBuff(3) As ChanelInf1
FileName = "TestData1"
BlockNo = 2
ret = File.HeaderReadS(FileName,BlockNo,ComBuff,ChBuff)
If 0 <> ret Then
   MsgBox "File read error"
End If
```

DataRead

Description

Reads the data from the data file by specifying the block number.

Syntax

DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As SByte) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As SByte) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As Byte) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As Byte) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As Short) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As Short) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As UInt16) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As UInt16) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As Integer) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As Integer) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As UInt32) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As UInt32) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As Single) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As Single) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*() As Double) As Short
DataRead(*FileName* As String,*BlockNo* As Integer,*ChNo* As Integer,*DataBuff*(,) As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
BlockNo (IN)	Block number.
ChNo (IN)	Channel number. -1 represents all channels. Counted from 1. If you specify -1, pass a buffer defined using a two-dimensional array for DataBuff.
DataBuff (OUT)	Data storage buffer. Pass the buffer in the type specified by DataForm.

Note:

Reads the data from the data file in units of blocks. The files cannot be handled as sequential files.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNo As Integer
Dim ChNo As Integer
Dim DataBuff(3,999) As Single
FileName = "TestData1"
BlockNo = 2
ChNo = -1
ret = File.DataRead(FileName,BlockNo,ChNo,DataBuff)
If 0 <> ret Then
   MsgBox "File read error"
End If
```

HeaderWriteS

Description

Writes the header information at once to the header file by specifying the block.

Syntax

HeaderWriteS(*FileName* As String,*BlockNo* As Integer,ByRef *ComBuff* As CommonInf1, *ChBuff*() As ChanelInf1, *AcqInfo*() As AcqDataInfEx2) As Short
HeaderWriteS(*FileName* As String,*BlockNo* As Integer,ByRef *ComBuff* As CommonInf1, *ChBuff*() As ChanelInf1, *AcqInfoEx*() As AcqDataInfEx) As Short (old interface)

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Header file name.
BlockNo (IN)	Block number. Specify the block number to be written.
ComBuff() (IN)	Common information structure. Define using an array.
ChBuff() (IN)	Channel information structure. Define using an array.
AcqInfo (IN)	Data information structure (AcqDataInfoEx2).
AcqInfoEx (IN)	Data information structure (AcqDataInfoEx).

Note:

Collectively writes the header file information for data of which the number of X-axis data points is the same. Prepare a single-element array for the common information structure buffer. Prepare channel information structure buffer for the amount equal to the number of channels. Pass the data obtained by GetAcqDataInfoEx() or AcqInfoInitialize() to AcqData. Data is written to the specified block. (The old interface does not have Date, Time, vUnit, and xUnit.)

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNo As Integer
Dim ComBuff As CommonInf1
Dim ChBuff(5) As ChanelInf1
Dim AcqInfo(5) As AcqDataInfoEx2
Dim InfoNum As Short
BlockNo=2
Ret = Module1.GetAcqDataInfo(-1,BlockNo,AcqInfo(),infoNum)
If 0 <> ret Then
   MsgBox "Data information read error"
EndIf
FileName = "TestData1"
ret = File.HeaderWriteS(FileName,BlockNo,ComBuff,ChBuff,AcqInfo)
If 0 <> ret Then
   MsgBox "File write error"
End If
```

DataWrite

Description

Writes the data to the data file in units of blocks.

Syntax

DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As SByte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As SByte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As Byte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As Byte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As Short) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As Short) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As UInt16) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As UInt16) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As Integer) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As Integer) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As UInt32) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As UInt32) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As Single) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As Single) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*() As Double) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, *DataBuff*(,) As Double) As Short

Old interface

DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As SByte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As SByte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As Byte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As Byte) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As Short) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As Short) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As UInt16) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As UInt16) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As Integer) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As Integer) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As UInt32) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As UInt32) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As Single) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As Single) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*() As Double) As Short
DataWrite(*FileName* As String,*BlockNo* As Integer, SampleNum As Integer, AcqInfo() As AcqDataInfoEx, *DataBuff*(,) As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension. A .wvf extension is added to the created file.
BlockNo (IN)	Block number.
DataBuff (OUT)	Data storage buffer.

Note:

Write data to a data file. The files cannot be handled as sequential files.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNo As Integer
Dim AcqInfo(1) As AcqDataInfoEx2
Dim DataBuff(999,1) As Single
Dim InfoNum As Integer
BlockNo = 2
Module1.GetAcqDataInfo(-1,BlockNo,AcqInfo(),InfoNum)
FileName = "TestData1"
ret = File.DataWrite(FileName,BlockNo,DataBuff)
If 0 <> ret Then
   MsgBox "File write error"
End If
```

HeaderCsReadS

Description

Collectively reads the header information from a header file.

Syntax

HeaderCsReadS(*FileName* As String,*SeriesNo* As Integer,ByRef *ComBuff* As CommonInf1, *ChBuff()* As ChanelInf1) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Header file name.
SeriesNo (IN)	Sequence number when creating sequence files. If you specify -1, FileName becomes the file name as-is.
ComBuff (OUT)	Common information structure.
ChBuff() (OUT)	Channel information structure. Define using an array.

Note:

Collectively retrieves the header file information for data of which the number of X-axis data points is the same. Prepare a single-element array for the common information structure buffer.
Prepare channel information structure buffer for the amount equal to the number of channels.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim SeriesNo As Integer
Dim ComBuff As CommonInf1
Dim ChBuff(5) As ChanelInf1
FileName = "TestData1"
SeriesNo = 2
ret = File.HeaderCsReadS(FileName,SeriesNo,ComBuff,ChBuff)
If 0 <> ret Then
   MsgBox "File read error"
End If
```

CsRead

Description

Reads the data from the data files (sequential files) by specifying the number of samples.

Syntax

CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As SByte) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As SByte) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As Byte) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As Byte) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As Short) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As Short) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As UInt16) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As UInt16) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* AsInteger, *DataBuff*() As Integer) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As Integer) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As UInt32) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As UInt32) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As Single) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As Single) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*() As Double) As Short
CsRead(*FileName* As String,*SeriesNo* As Integer,*Start* As Integer,*Length* As Integer, *ChNo* As Integer, *DataBuff*(,) As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
SeriesNo (IN)	Sequence number when reading sequential files. If you specify -1, only the file with the name specified by FileName is read.
Start (IN)	Start sample number. Specify the first sample number to be retrieved using a value greater than or equal to 0.
Length (IN)	Number of samples to be retrieved. -1 specifies all data after the sample specified by Start. An error occurs if the specified value is greater than the number of samples that is stored.
ChNo (IN)	Channel number. One origin. -1 specifies all channels.
DataBuff (OUT)	Buffer for storing data. Pass the buffer in the type specified by DataForm.

Note:

Reads the data from the data file by specifying the number of samples.
Retrieves data of Length from sample Start in the file specified by FileName and SeriesNo.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim SeriesNo As Integer
Dim Start As Integer
Dim Length As Integer
Dim ChNo As Integer
Dim DataBuff(3,1199) As Single
FileName = "TestData1"
SeriesNo = 2
Start = 100
Length = 1200
ChNo = -1
ret = File.CsRead(FileName,SeriesNo,Start,Length,ChNo,DataBuff)
If 0 <> ret Then
   MsgBox "File read error"
End If
```

HeaderCsWriteS

Description

Collectively writes the header information to the header file.

Syntax

HeaderCsWriteS(*FileName* As String,*SeriesNo* As Integer , ByRef *ComBuff* As CommonInf1,*ChBuff*() As ChanelInf1,*AcqInfo*() As AcqDataInfoEx2)
HeaderCsWriteS(*FileName* As String,*SeriesNo* As Integer , ByRef *ComBuff* As CommonInf1,*ChBuff*() As ChanelInf1,*AcqInfo*() As AcqDataInfoEx) (old interface)

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Header file name.
SeriesNo (IN)	Sequence number when creating sequence files. If you specify -1, FileName becomes the file name as-is.
CommBuff (IN)	Common information structure.
ChBuff() (IN)	Channel information structure. Define using an array.
AcqInfo (IN)	Data information structure. Define using an array.

Note:

Collectively writes the header file information for data of which the number of X-axis data points is the same. Prepare a single-element array for the common information structure buffer.
Prepare channel information structure buffer for the amount equal to the number of channels.
Pass the data obtained by GetAcqDataInfo() or AcqInfoInitialize() to AcqData.
If the file already exists, it is overwritten.

Example (Visual Basic .Net)

```
Dim ret As Long
Dim FileName As String
Dim SeriesNo As Integer
Dim CommonBuff As CommonInf1
Dim ChBuff(5) As ChanelInf1
Dim AcqInfo(5) As AcqDataInfoEx2
Dim InfoNum As Long
Ret = Module1.GetAcqDataInfo(-1,5,AcqInfo,infoNum)
If 0 <> ret Then
   MsgBox "Data information read error"
End If
FileName = "TestData1"
SeriesNo = 2
ret = File.HeaderCsWriteS(FileName,SeriesNo,CommonBuff,ChBuff,AcqInfo)
If 0 <> ret Then
   MsgBox "File write error"
End If
```

CsWrite

Description

Write data to a sequence file.

Syntax

CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As SByte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As SByte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As Byte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As Byte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As Short) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As Short) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As UInt16) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As UInt16) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As Integer) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As Integer) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As UInt32) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As UInt32) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As Single) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As Single) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*() As Double) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer, *DataBuff*(,) As Double) As Short

Old interface

CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As SByte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As SByte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As Byte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Byte) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As Short) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Short) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As UInt16) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Uint16) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As Integer) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Integer) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As UInt32) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As UInt32) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As Single) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Single) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*() As Double) As Short
CsWrite(*FileName* As String,*SeriesNo* As Integer,*SampleNum* As Integer, *AcqInfo*() As AcqDataInfoEx, *DataBuff*(,) As Double) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension. A .wvf extension is added to the created file.
SeriesNo (IN)	Sequence number when creating sequence files. If you specify -1, FileName becomes the file name as-is.
SampleNum (IN)	Number of samples.
AcqInfo (IN)	Data information structure. Pass the data that has been returned by GetAcqDataInfo() or DPInitializeAcqInfo().
DataBuff (OUT)	Data storage buffer.

Note:

Write data to a data file.
The files are handled as sequential files. SeriesNo is automatically added to the file names.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim SeriesNo As integer
Dim DataBuff(1,1000) As Single
FileName = "TestData1"
SeriesNo = 2
ret = File.CsWrite(FileName, SeriesNo,DataBuff)
If 0 <> ret Then
   MsgBox "File write error"
End If
```

HeaderItemRead

Description

Reads the information of the specified item name and specified channel from the header information of the header file.

Syntax

HeaderItemRead(*FileName* As String, *ItemName* As String, *ChNo* As Integer, *BlockNo* As Integer, ByRef *DataBuff* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
ItemName (IN)	Item name.
ChNo (IN)	Channel number. Ignored if an item unrelated to the channel number is specified.
BlockNo (IN)	Block number.
DataBuff (OUT)	Buffer for storing data.

Note:

Retrieves the information of the specified item and specified channel from the header file information.
You must have an understanding of the structure of the .hdr file when using this function.
A block number is added to the item number for data containing multiple blocks. To read such item, specify the item name with the block number.
Items “GroupNumber” and “TraceNumber” cannot be retrieved.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim ItemName As String
Dim ChNo As Integer
Dim DataBuff As String
Dim BlockNo As Integer
FileName = "TestData"
ItemName = "VResolution"
ChNo = 1
BlockNo = 0
Ret = File.HeaderItemRead(FileName,ItemName,ChNo,BlockNo,DataBuff)
If 0 <> ret Then
   MsgBox "Error in reading a header file item"
End If
```

HeaderItemWrite

Description

Writes data to the specified item name and specified channel in the header information of the header file.

Syntax

HeaderItemWrite(*FileName* As String, *ItemName* As String, *ChNo* As Integer, *BlockNo* As Integer, *DataBuff* As String) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
ItemName (IN)	Item name.
ChNo (IN)	Channel number.
BlockNo (IN)	Block number.
DataBuff (OUT)	Buffer for storing data.

Note:

Writes data to the specified item and specified channel in the header information file.
Items “FormatVersion,” “Model,” “Endian,” “DataFormat,” “GroupNumber,” “TraceTotalNumber,” “TraceName,” “BlockNumber,” and “VDataType” cannot be specified.
When you set PLinearMode, set to “0” as OFF or “1” as ON.
If PlinearMode is set to 1, the data value is displayed using a value converted by the Linear scale value when the corresponding file is viewed on the Control Software Viewer.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim ItemName As String
Dim ChNo As Integer
Dim BlockNo As Integer
Dim DataBuff As String
FileName = "TestData"
ItemName = "VResolution"
ChNo = 1
BlockNo = 0
DataBuff = CStr(5.42)
Ret = File.HeaderItemWrite(FileName,ItemName,ChNo,BlockNo,DataBuff)
If 0 <> ret Then
   MsgBox "Error in reading a header file item"
End If
```

GetSampleChNum

Description

Gets the number of samples and number of channels of the specified file.

Syntax

GetSampleChNum(*FileName* As String, *BlockNo* As Integer, ByRef *SampleNum* As Integer,ByRef *ChNum* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
BlockNo (IN)	Block number. -1 specifies all blocks.
SampleNum (OUT)	Number of samples.
ChNum (OUT)	Number of channels.

Note:

The number of samples and number of channels of the specified file are returned.
For Scan type files, the total number of samples is returned regardless of the BlockNo setting.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNo As Integer
Dim SampleNum As Integer
Dim ChNum As Integer
FileName = "TestData"
BlockNo = 0
ret = File.GetSampleChNum(FileName,SampleNum,ChNum)
If 0 <> ret Then
   MsgBox "Error in reading the number of samples and number of channels"
End If
```

GetBlockNum

Description

Gets the number of blocks of the specified file.

Syntax

GetBlockNum(*FileName* As String,ByRef *BlockNum* As Integer) As Short

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

FileName (IN)	Data file name. Specify the file name excluding the extension.
BlockNum (OUT)	Number of blocks.

Note:

The number of blocks of the specified file is returned.

Example (Visual Basic .Net)

```
Dim FileName As String
Dim BlockNum As Long
FileName = "TestData"
ret = File.GetBlockNum(FileName,BlockNum)
If 0 <> ret Then
   MsgBox "Error in reading the number of blocks"
End If
```

InitializeAcqInfo

Description

Stores the required data in the data information structure.

Syntax

InitializeAcqInfo(*VMaxData* AsDouble, *VMinData* As Double, *SampleNum* As Integer, *SampInterval* As Double, *AcqInfo()* As AcqDataInfoEx2)

(Old interface)

InitializeAcqInfo(*VMaxData* AsDouble, *VMinData* As Double, *SampleNum* As Integer, *SampInterval* As Double, *AcqInfoEx()* As AcqDataInfoEx)

Return value

Returns 0 if successful. Returns an error code if unsuccessful.

Parameters

VMaxData (IN)	Scale Max.
VMinData (IN)	Scale Min.
SampleNum (IN)	Number of samples.
SampInterval (IN)	Sampling interval (s).
AcqInfo() (OUT)	Data information structure.
AcqInfoEx() (OUT)	Data information structure. (for old interface)

Note:

Sets and returns the required data in the data information structure.
VMaxData and VMinData are the scale values of the Y-axis when the measured data in the stored file is displayed using the Waveform Monitor of the WE7000 Control Software.
The data set in AcqDataInfo by this function is as follows:

Chanel	1 through n is set in the order of the array.
dataType	Data buffer type is set using the parameter of DataWrite or CsWrite.
blockNum	1
startBit	0
effectiveBit	0
trigActive	0
record	SampleNum.
recordLen	SampleNum.
time	0
trigPosition	0.0
Interval	Set to the SamplingInterval parameter.
VResolution	Set to 1.0.
VOffset	Set to 0.0.
TrigLevel	Set to 0.0.
TrigWidth	Set to 0.0.
PlusOverData	Set to the VMaxData parameter.
MinusOverData	Set to the VMinData parameter.
NonData	Set to the lost data.
DispMaxData	Set to the VMaxData parameter.
DispMinData	Set to the VMinData parameter.
Date	Date when this function was called (do not set anything on the old interface.)
Time	Time when this function was called (do not set anything on the old interface.)
Vunit	Sets “V” (do not set anything on the old interface.)
Xunit	Sets “See” (do not set anything on the old interface.)

Example (Visual Basic .Net)

```
Dim VMaxData As Double
Dim VMinData As Double
Dim AcqInfo(3) As AcqDataInfoEx2
VMaxData = 2.0
VMinData = –2.0
ret = File.InitializeAcqInfo(VMaxData,VMinData,1000,0.001,AcqInfo)
If 0 <> ret Then
   MsgBox "Error in setting data to data information structure"
End If
```

4.1 Error Codes

WeControl, WeStation, WeModule Class

Controller Side

Error Code (Hexadecimal)	Description
1 to 19 (0x1 to x13)	Communication driver error codes.
100 (0x64)	Failed initialization.
101 (0x65)	Buffer is full.
102 (0x66)	No communication interface board.
103 (0x67)	Station does not exist.
104 (0x68)	Event receive error.
105 (0x69)	Abnormal interrupt.
106 (0x6A)	Abnormal received packet.
107 (0x6B)	Buffer size too small.
110 (0x6E)	Requested reply packet not received.
111 (0x6F)	RplySyncExec API error.
112 (0x70)	SettingQuery API error.
113 (0x71)	Cannot process segmented packets.
114 (0x72)	No such function.
115 (0x73)	Not all segmented packets have been received.
200 (0xC8)	Execution error on station.
201 (0xC9)	Execution error on module.
202 (0xCA)	No station name.
203 (0xCB)	Duplicate station names.
204 (0xCC)	Bad station name.
205 (0xCD)	Module does not exist.
206 (0xCE)	Bad module name.
207 (0xCF)	Invalid slot specification.
300 (0x12C)	Function not supported.
301 (0x12D)	Function not supported.
302 (0x12E)	Acquisition data does not exist.
400 (0x190)	Failed to allocate dynamic memory.
401 (0x191)	Failed in sending messages between threads.
402 (0x192)	Failed to create thread.
403 (0x193)	File open error.
404 (0x194)	File read/write error.
405 (0x195)	File access error.
500 (0x1F4)	Handle error.
501 (0x1F5)	ASCII command cannot be found.
502 (0x1F6)	Bad parameter.
503 (0x1F7)	Cannot open the module handle of a child.
504 (0x1F8)	Invalid setting.

Common Error Codes for the Station and Modules

Error Code (Hexadecimal)	Description
4097 (0x1001)	Parameter unnecessary.
4098 (0x1002)	Value was limited.
4099 (0x1003)	Over the range.
4100 (0x1004)	Lost preset information.
8193 (0x2001)	Bad module ID.
8194 (0x2002)	Bad control ID.
8195 (0x2003)	Bad command ID.
8196 (0x2004)	Parameter necessary.
8197 (0x2005)	Parameter type different.
8198 (0x2006)	Too many parameters.
8199 (0x2007)	Not enough parameters.
8200 (0x2008)	Setting conflict.
8201 (0x2009)	Hardware not installed.
8202 (0x200A)	Program error.
8203 (0x200B)	Lost calibration value.
8204 (0x200C)	Failed self test.
8205 (0x200D)	Failed calibration.
8206 (0x200E)	Data exceeds the range.
8207 (0x200F)	Failed measurement.
8208 (0x2010)	Hardware error.
8209 (0x2011)	Temperature error.
8210 (0x2012)	Cooling fan stopped.
8211 (0x2013)	System memory overflow.
8212 (0x2014)	Bad data base format.
8213 (0x2015)	Bad data base version.
8214 (0x2016)	Bad handle was provided.
8215 (0x2017)	Failed to get handle.
8216 (0x2018)	Measurement aborted.
8217 (0x2019)	No measurement data.
8218 (0x201A)	Timeout.
8219 (0x201B)	Data overrun occurred.
8225 to 8232 (0x202#)	Program error in SLOT# (#: Slot number 1 to 8).
8241 to 8248 (0x203#)	Bad version in SLOT# (#: Slot number 1 to 8).

WeFilter Class

Symbol	Error Code (Hexadecimal)	Description
AsciiRSLT_NORMAL	0 (0x0000)	Normal termination.
AsciiRSLT_CANT_OPEN	1 (0x0001)	Failed to open file.
AsciiRSLT_CANT_ALLOC	2 (0x0002)	Failed to allocate memory.
AsciiRSLT_CANT_READ	9 (0x0009)	Failed to retrieve data (includes “?,” for example).Vresolution, Voffset, VUnit, VPlusOverData, VminusOverData, VillegalData, VMaxData, VMinData, Hresolution, HUnit, Data, Time.
AsciiRSLT_BOUNDARY	10 (0x000A)	Boundary specification error.
AsciiRSLT_READ_ERROR	11 (0x000B)	Access error to the wvf file.
AsciiRSLT_LINK_ERROR	13 (0x000D)	Failed to link to the DLL.
AsciiRSLT_UNSUPPORTED_FUNCTION	14 (0x000E)	Function not supported by this DLL.
AsciiRSLT_ZERO_BLOCKSIZE	15 (0x000F)	Block size is zero.
AsciiRSLT_ERROR	255 (0x00FF)	Other error.

WeFile Class

Symbol	Error Code (Hexadecimal)	Description
AsciiRSLT_NORMAL	0x0000	Normal completion.
AsciiRSLT_CANT_OPEN	0x0001	File open failure.
AsciiRSLT_CANT_ALLOC	0x0002	Memory allocation failure.
AsciiRSLT_FORMAT_ERROR	0x0003	The number of fields of the record is not correct.
AsciiRSLT_HANDLE_NULL	0x0004	NULL handle.
AsciiRSLT_NOT_INTEGER	0x0005	The data type is not UINT. GroupNumber, TraceTotalNumber, DataOffset, TraceNumber, BlockNumber, and BlockSize are UINT.
AsciiRSLT_POSITION_ERROR	0x0006	Description position is not correct. For example, a Group item is in PublicInfo.
AsciiRSLT_SCRIPT_ERROR	0x0007	Error in the item descriptor. For example, BlockSizeA.
AsciiRSLT_STRING_ERROR	0x0008	Error in Endian or DataFormat. Endian: Little or Big DataFormat: Trace or Block
AsciiRSLT_CANT_READ	0x0009	Failed to read. For example, “?”. Vresolution, Voffset, VUnit, VPlusOverData, VminusOverData, VillegalData, VMaxData, VMinData, Hresolution, HUnit, Date, or Time
AsciiRSLT_BOUNDARY	0x000A	Range designation error.
AsciiRSLT_READ_ERROR	0x000B	Access error to WVF file.
AsciiRSLT_UNSUPPORTED_DATATYPE	0x000C	Error in DataType. Only IU1, IS1, IU2, IS2, IU4, IS4, FS4, and FS8 are supported.
AsciiRSLT_LINK_ERROR	0x000D	DLL link failure.
AsciiRSLT_UNSUPPORTED_FUNCTION	0x000E	Function not supported by the specified DLL.
AsciiRSLT_ZERO_BLOCKSIZE	0x000F	The block size is 0.
AsciiRSLT_UNSUPPORTED_SCALING	0x0010	Scaling designation error. Other than AsciiADWave, AsciiPhysicalWave, or AsciiScalingWave.
AsciiRSLT_DISK_FULL	0x0011	Insufficient free disk space.
AsciiRSLT_MODIFY_ERROR	0x0012	Error in the parameter.
AsciiRSLT_ERROR	0x00FF	Other error.

Index

S

T

W