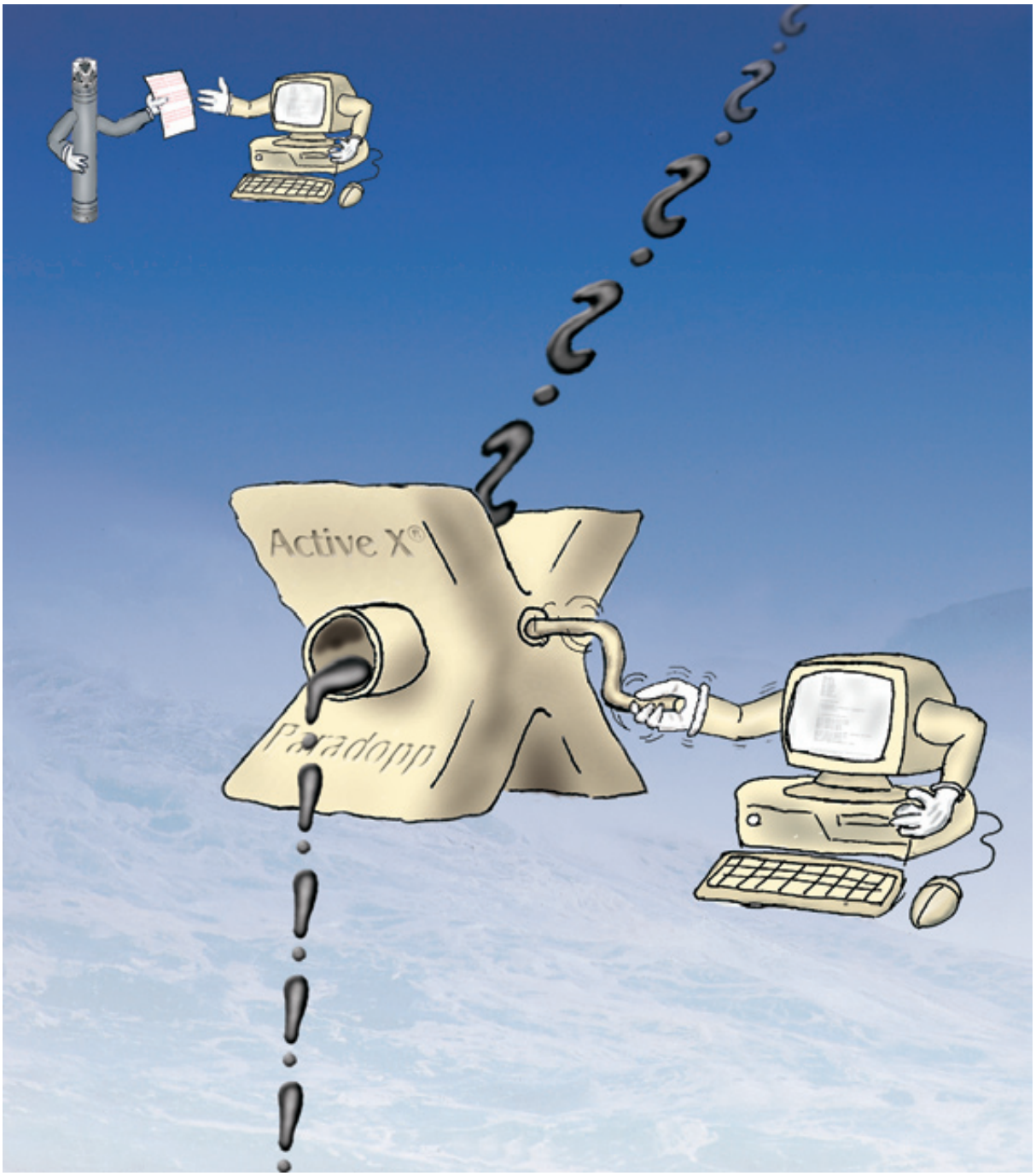
NORTEK AS

ACTIVEX®

MODULE FOR SYSTEM INTEGRATORS



PARADOPP FAMILY OF PRODUCTS

MAY 2004

N3009-104 • Rev. B

ACTIVEX®

MODULE FOR SYSTEM INTEGRATORS

PARADOPP FAMILY OF PRODUCTS

Nortek AS, Vangkroken 2, NO-1351 RUD, Norway.

Tel: +47 67 17 45 00 • Fax: +47 67 13 67 70 • e-mail: inquiry@nortek.no • www.nortek-as.com

NORTEK AS

Software updates and technical support

Find us on the world wide web:

www.nortek-as.com, www.nortek.no

Here you will find software updates and technical support.

Your Feedback is appreciated

If you find errors, misspelled words, omissions or sections poorly explained, please do not hesitate to contact us and tell us about it at:

inquiry@nortek.no

We appreciate your comments and your fellow users will as well.

Nortek Forum Support

If you have comments, application tips, suggestions to improvements, etc. that you think will be of general interest you should register on Nortek's Forums at

www.nortek-as.com/cgi-bin/ib/ikonboard.cgi

and post your message there. The Forums also offer a great opportunity to share your experience using Nortek sensors with other users around the world, and to learn from their experience.

Communicating with us

If you need more information, support or other assistance from us, do not hesitate to contact us:

Nortek AS
Vangkroken 2
NO-1351 RUD, Norway
Phone: +47 6717 4500, Fax: +47 6713 6770
e-mail: inquiry@nortek.no

OVERVIEW

What's in this Manual?

Chapter 1 – Introduction

Here we introduce you to the ActiveX® documentation, describes the installation procedure, and provides an overivew of the technology.

Chapter 2 – Implementing the Technology

This chapter provides descriptions of the principles of coding for the most popular programming languages supporting OLE automation.

Chapter 3 – Function Reference

This chapter provides a list of all the functions available, sorted alphabetically.

Chapter 4 – Implementation Examples

Implementation examples are provided in Visual C++®, Visual Basic® and Delphi™.

DETAILS

The Table of Contents

CHAPTER 1
Introduction

This document provides the information necessary to control a Nortek Paradopp product (Aquadopp, Vector, etc.) using the PdCommX ActiveX® control function interface. The document is aimed at system integrators and engineers in need of writing their own Win32® or Windows® CE applications to control one or more Paradopp products. Examples are provided in Visual C++®, Visual Basic® and Delphi™. The scope is limited to interfacing issues and the document does not address the general performance issues of the systems. For a more thorough understanding of the principle of operation, please consult the User Guide for the instrument in question.

Installation

Install the PdCommX component on your computer by using the program **setup.exe**. The setup program copies files from the CD onto your harddisk and adds the necessary interface information to the Windows® registry.

For Windows® CE installations, a link between your mobile device and your desktop computer must be established using Microsoft® ActiveSync® before you run the setup.

Overview of the Technology

ActiveX® controls technology builds on a foundation of many lower-level objects and interfaces in OLE Automation. An external application accessing an Automation object does so by first connecting to the object and then requesting access to one or more of its published interfaces. An interface is an entry point that allows access to one or more related methods, properties and events. Once

an application obtains an interface on the object, it proceeds to call any interface methods, as though they were part of the application itself.

Think of the PdCommX component as a programmable replacement for the standard Paradopp software user interface – the behaviour seen when using the standard software interface is to a large extent the same behaviour seen when using the ActiveX control. The PdCommX properties correspond to deployment planning and configuration dialog parameters and the PdCommX methods correspond to menu commands. A PdCommX event is generated whenever new data is available from the instrument.

CHAPTER 2
Implementing the Technology

When connected to the Automation object and its interfaces, identifier parameters, also known as Globally Unique Identifier (GUIDs), are passed to the automation library API functions. There is a single GUID representing the object itself and a GUID for each of the exposed interfaces. In order for the application to succeed in connecting to the object and its interfaces, these GUIDs must be present in the registry. The GUID entries are added to the registry by the PdCommX installation program.

The PdCommX control may be accessed by any language platform or software development environment (SDE) that supports OLE Automation (OCX interface). The most popular such languages platforms are Microsoft Visual C++®, Microsoft Visual Basic® and Borland Delphi™. The principles for coding for these platforms is similar, although in the Visual Basic® environment much of the low level work is performed behind the scenes by the Visual Basic® run-time system.

A general PdCommX OCX implementation is as follows:

1. Register **PdCommX.ocx** with the operating system (This is done by the PdCommX setup, however, most SDE's provide tools to register ActiveX® controls)
2. Include the **PdCommX.ocx** file into your application project.
3. Place the **PdCommX** control onto a destination form.
4. Use the property window to adjust default properties as desired.
5. Adjust properties in code as desired (Typically in the Visual Basic® **Form_Load** event or in the Delphi™ **FormCreate** procedure)
6. Implement the PdCommX methods and a **NewData** event handler as desired.

Note: The **Connect** method must be called prior to any other methods. Also, in order for some of the properties settings to take effect, the **SetConfig** method must be called prior to the **Start** command.

What follows is a brief overview of the programming techniques that allow easy access to the PdCommX interface from either of these languages.

Using Microsoft Visual C++®:

1. Create an MFC application using the **MFC AppWizard** (Ensure the ActiveX® Controls box is checked in step 3).
2. Open the **Components and Controls** gallery
3. Choose **PdCommX** in the **Registered ActiveX®** controls folder.
4. Click **Insert** to generate the CPdCommX class files.
5. Place the PdCommX control onto a dialogue (dialogue-based application) or a form view (form-based application)

Using Microsoft Visual Basic®:

1. Create a Standard EXE project.
2. Open the **Components** dialogue.
3. Check the **PdCommX ActiveX®** Control module in the **Controls** tab list.
4. Click **OK** to add the PdCommX control to the Toolbox.
5. Place the PdCommX control onto a form.

Using Borland Delphi™:

1. Create a new **Application** project.
2. Open the **Import ActiveX®** dialogue.
3. Select the **PdCommX ActiveX®** Control module in the controls list.
4. Click **Install** to create a PdCommX class unit and add the PdCommX control to your project.
5. Place the PdCommX control onto a form.

CHAPTER 3
Function Reference

Properties

The following methods are used to get and set properties of the PdCommX object. Since each instrument type has specific features, some properties only apply to one or some of the instruments. The **Scope** heading in the property reference identifies the instrument type.

The instrument operation parameters may be specified at two levels. The **Standard** level provides default settings for various environments and mounting arrangements. The **Advanced** level allows for fine tuning the operation parameters. The **Category** heading in the reference specifies the level. Note that the **ConfigLevel** property must be set to **Advanced** for the property settings categorized as Advanced to be effective.

The description includes function definitions in the Microsoft Visual C++® language. For function definition details (return types and parameter types) in other languages use your SDE object browser.

Property settings in the **Communication** category control the serial communication and must be set prior to the the **Connect** method to be effective.

Also note that instrument operation parameters (Standard and Advanced) are neither written to nor saved in the instrument until the **SetConfig** method is called. This means that when you are in software design mode, the SDE object property window will not reflect the instrument settings.

AnalogInput1

Scope	Description
All except Vectrino	The instrument can read two analogue inputs at the same time. The input range is 0-5 Volt, where 0 Volt equals 0 counts, 5 Volts equals 65535 counts and 2.5 Volts equals 32768 counts. The voltage output is fixed in production to either 5 Volts, 12 Volts or to the instrument voltage. The use of analogue inputs requires a special internal harness. Some systems are equipped with this at the time of purchase. It is also possible to purchase the harness separately and upgrade the instrument. Values are 0 = None, 1 = Standard(Profile), 2 = Fast(Wave), 3 = StandardFast(Profile & Wave).
Category	**Visual C++®**
Advanced	`short GetAnalogInput1();` `void SetAnalogInput1(short nNewValue);`
Default	
0 (None)	

AnalogInput2

Scope	Description
All except Vectrino	See AnalogInput1.
Category	**Visual C++®**
Advanced	`short GetAnalogInput2();` `void SetAnalogInput2(short nNewValue);`
Default	
0 (None)	

AnalogOutput

Scope	Description
Vector, Vectrino	Enables or disables analogue output. When enabled, the 3D velocity is output as 0-5 Volt continuous signal over a separate set of three wires, one for each velocity component. Values are 0 = OFF, 1 = ON.
Category	**Visual C++®**
Advanced (Vector) Standard (Vectrino)	`short GetAnalogOutput();` `void SetAnalogOutput(short nNewValue);`
Default	
0 (OFF)	

AnalogRange

Scope	Description
Vector, Vectrino	AnalogRange specifies the velocity range that will correspond to the full analogue output range. Values are 0 = 7.0 m/s, 1 = 2.5 m/s, 2 = 1.0 m/s, 3 = 0.3 m/s.

Category
Advanced (Vector)
Standard (Vectrino)
	Visual C++®
	`short GetAnalogRange();`
	`void SetAnalogRange(short nNewValue);`

Default 3 (0.3 m/s)	

AutoConnect

Scope	Description
All instruments	Enables or disables automatic baud rate scanning. Values are 0 = OFF, 1 = ON.

Category
Communication
	Visual C++®
	`short GetAutoConnect();`
	`void SetAutoConnect(short nNewValue);`

Default 0 (OFF)	

AvgInterval

Scope	Description
Aquadopp, Aquapro, EasyQ, AWAC, Continental	Averaging Interval specifies how long the instrument will be actively collecting data within each Measurement Interval. Example: Measurement Interval is set to 600 seconds, averaging interval set to 60 seconds means that the instrument will collect data for 1 minute every 10 minutes. Averaging Interval must be smaller than Measurement Interval.

Category
Standard (EasyQ)
Advanced (all others)
	Visual C++®
	`short GetAvgInterval();`
	`void SetAvgInterval(short nNewValue);`

Default	

BaudRate

Scope	Description
All instruments	The baud rate to use for the connection. Values are 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

Category
Communication
	Visual C++®
	`long GetBaudRate();`
	`void SetBaudRate(long nNewValue);`

Default 9600	

BlankingDistance	
Scope Aquadopp, Aquapro, EasyQ, AWAC, Continental	Description Blanking distance is the distance in metres from the Aquadopp sensor head to the start of the (first) measurement cell.

Category
Standard (EasyQ)
Advanced (all others)
	Visual C++®
	`float GetBlankingDistance();`
	`void SetBlankingDistance(float newValue);`

Default	

BreakType	
Scope All instruments	Description Sets instrument break command type. Values are 0 = Hard Break, 1 = Soft Break. Which break type to use depends on the production date of the Nortek instrument. Most instruments manufactured after 2001 use soft break. Run the software shipped with the instrument to find out which type of break to use.
Category Communication	

Default
1 (Soft Break)
	Visual C++®
	`short GetBreakType();`
	`void SetBreakType(short nNewValue);`

CellSize	
Scope Aquapro, AWAC, Continental	Description Cell size specifies the vertical length in metres of each depth cell in the profile. Applies to profiling instruments only. Standard deviation (accuracy) of the velocity measurements is inversely proportional to the depth cell size (larger depth cells give smaller standard deviations).
Category Standard	

Default	Visual C++®
	`float GetCellSize();`
	`void SetCellSize(float newValue);`

CompassUpdateRate

Scope	Description
All, except Vector, Vectrino and EasyQ	Compass update rate sets how often the compass data is collected. The maximum rate is once every second and the minimum rate is once every averaging interval.
Category	**Visual C++®**
Advanced	`short GetCompassUpdateRate();` `void SetCompassUpdateRate(short nNewValue);`
Default	

CoordinateSystem

Scope	Description
All	The coordinate system can be selected to Beam, XYZ, or ENU. Beam means that the recorded velocity will be in the coordinate system of the acoustic beams. XYZ means that the measurements are transformed to an orthogonal XYZ coordinate system fixed relative to the instrument and ENU means that the data are converted to geographic coordinates every second. Values are 0 = ENU, 1 = XYZ, 2 = Beam. The use of ENU requires that the instrument is equipped with a compass.
Category	**Visual C++®**
Standard (Vectrino) Advanced (all others)	`short GetCoordinateSystem();` `void SetCoordinateSystem(short nNewValue);`
Default	
0 (ENU)	

ConfigLevel

Scope	Description
All instruments	Properties controlling the instrument may be specified at two levels. Use the Standard level to configure the system with default settings for various environments and mounting arrangements. Use the Advanced level to fine tune the operation parameters. Values are 0 = Standard, 1 = Advanced.
Category	**Visual C++®**
Application	`short GetConfigLevel();` `void SetConfigLevel(short nNewValue);`
Default	
0 (Standard)	

DeploymentComment	
Scope All, except Vectrino	Description Max 180 characters of text which will be included in the data file for the purpose of documenting the data set.

Category
Deployment
	Visual C++®
	`BSTR GetDeploymentComment();`
	`void SetDeploymentComment(LPCTSTR lpszNewValue);`

Default “N/A”	

DeploymentName	
Scope All, except Vectrino	Description Recorder data file name. Max 6 characters of text.

Category
Deployment
	Visual C++®
	`BSTR GetDeploymentName();`
	`void SetDeploymentName(LPCTSTR lpszNewValue);`

Default “N/A”	

DeploymentTime	
Scope All, except Vectrino	Description Sets date and time to start data recording.

Category
Deployment
	Visual C++®
	`DATE GetDeploymentTime();`
	`void SetDeploymentTime(DATE newValue);`

Default Current time	

DiagnosticsInterval	
Scope Aquadopp, EasyQ	Description Sets the Interval in seconds for diagnostics data.

Category
Advanced
	Visual C++®
	`short GetDiagnosticsInterval();`
	`void SetDiagnosticsInterval(short nNewValue);`

Default	

DiagnosticsMode

Scope	Description
Aquadopp, EasyQ	Enables or disables diagnostics. If enabled, the diagnostic mode will set the instrument to collect a number of diagnostics samples at regular intervals. For example, if the interval is set to 720 and number of samples is set to 30, the instrument will record 30 1-second data points ever 12 hours. The number of samples cannot be larger than the Measurement interval minus the Averaging interval. Values are 0 = OFF, 1 = ON.
Category	
Advanced	
Default	**Visual C++®**
0 (OFF)	`short GetDiagnosticsMode();` `void SetDiagnosticsMode(short nNewValue);`

DiagnosticsSamples

Scope	Description
Aquadopp, EasyQ	Sets the number of diagnostics samples.
Category	**Visual C++®**
Advanced	`short GetDiagnosticsSamples();` `void SetDiagnosticsSamples(short nNewValue);`
Default	
20	

FileWrapping

Scope	Description
All, except Vectrino	Enables or disables recorder file wrapping. If enabled, data is logged to the internal instrument recorder in ring-buffer mode. This ensures that the recorder always holds the latest data. If disabled data logging will stop when the recorder is full. Values are 0 = OFF, 1 = ON.
Category	
Advanced	
Default	**Visual C++®**
0 (OFF)	`short GetFileWrapping();` `void SetFileWrapping(short nNewValue);`

HighRate

Scope	Description
Aquadopp	Enables or disables 4 Hz sampling rate. Values are 0 = OFF, 1 = ON.
Category	
Advanced	
Default	**Visual C++®**
0 (OFF)	`short GetHighRate();` `void SetHighRate(short nNewValue);`

MaxDepth	
Scope EasyQ	Description The estimated maximum depth (meters) is used to determine the transmit level for stage measurements. For direct control, set ConfigLevel = Advanced and use the advanced properties.

Category
Standard
	Visual C++®
	`float GetMaxDepth();`
	`void SetMaxDepth(float newValue);`

Default 10	

MeasInterval	
Scope All, except Vectrino	Description Measurement Interval describes how often the instrument will collect data and the value can be set from 1 to 3600 seconds. **Example:** Measurement Interval = 600 seconds means that data will be written to the recorder and/or to disk file every 10 minutes.

Category
Standard
	Visual C++®
	`short GetMeasInterval();`
	`void SetMeasInterval(short nNewValue);`

Default 600	

MeasLoad	
Scope All, except Vectrino	Description Within each second, the instrument can either be in active mode (collecting data) or in idle mode (not collecting data). The Measurement load is the relative time spent in active mode within each second and can have value from 0 (no data collection) to 100 (always in active mode).

Category
Advanced
	Visual C++®
	`short GetMeasLoad();`
	`void SetMeasLoad(short nNewValue);`

Default	

MeasLoadMode	
Scope All, except Vectrino	Description Enables or disables automatic calculation of an appropriate measurement load based on the averaging interval. Values are 0 = Manual, 1 = Automatic. See MeasLoad below.

Category
Advanced
	Visual C++®
	`short GetMeasLoadMode();`
	`void SetMeasLoadMode(short nNewValue);`

Default 1 (Automatic)	

Messages	
Scope All instruments	Description Enables or disables pop-up message boxes. Values are 0 = OFF, 1 = ON.

Category
Application
	Visual C++®
	`short GetMessages();`
	`void SetMessages(short nNewValue);`

Default 1 (ON)	

ModemConfig	
Scope All instruments	Description Modem AT configuration string.

Category
Communication
	Visual C++®
	`BSTR GetModemConfig();`
	`void SetModemConfig(LPCTSTR lpszNewValue);`

Default ATM0&D2	

ModemTimeout	
Scope All instruments	Description Modem response timeout in seconds.

Category
Communication
	Visual C++®
	`short GetModemTimeout();`
	`void SetModemTimeout(short nNewValue);`

Default 60	

NumCells	
Scope Aquapro, AWAC, Continental	Description Number of depth cells to collect per profile. Applies to profiling instruments only. The maximum number of depth cells with accurate velocity data will vary with system frequency, depth cell size and deployment conditions.

Category
Standard
	Visual C++®
	`short GetNumCells();`
	`void SetNumCells(short nNewValue);`

Default	

PhoneNumber	
Scope All instruments	Description Modem phone number to dial.

Category
Communication
	Visual C++®
	`BSTR GetPhoneNumber();`
	`void SetPhoneNumber(LPCTSTR lpszNewValue);`

Default <blank>	

PowerLevel	
Scope All	Description The power level sets how much acoustic energy the instrument transmits into the water. The difference between the highest level and the lowest level is about 20dB. Values are High = 0, HighLow = 1, LowHigh = 2, Low = 3.

Category
Advanced
	Visual C++®
	`short GetPowerLevel();`
	`void SetPowerLevel(short nNewValue);`

Default	

PowerOutMode	
Scope All, except Vectrino	Description Enable to supply power from the instrument to an external sensor. Values are 0 = OFF, 1 = ON.

Category
Advanced
	Visual C++®
	`short GetPowerOutMode();`
	`void SetPowerOutMode(short nNewValue);`

Default 0 (Off)	

PowerSource	
Scope EasyQ	Description Sets the instrument power source. The power source determines the default EasyQ transmit level (flow measurements). For direct control, set ConfigLevel = Advanced and use the advanced properties. Values are 0 = Battery, 1 = External.

Category
Standard
	Visual C++®
	`short GetPowerSource();`
	`void SetPowerSource(short nNewValue);`

Default 0 (Battery)	

QualityThreshold	
Scope EasyQ	Description The stage algorithm looks for the first echo peak for which its quality crosses above this threshold.

Category
Standard
	Visual C++®
	`short GetQualityThreshold();`
	`void SetQualityThreshold(short nNewValue);`

Default 150	

Salinity	
Scope All	Description Salinity to use for sound speed calculation. The salinity is 0 for fresh water and typically 35 for the ocean.

Category
Standard (Vector/Vectrino)
Advanced (all others)
	Visual C++®
	`float GetSalinity();`
	`void SetSalinity(float newValue);`

Default 35.0	

SampleOnSynch	
Scope Vector/Vectrino	Description Enables or disables start data collection by external trigger signal.Values are 0 = OFF, 1 = ON.

Category
Advanced (Vector)
Standard (Vectrino)
	Visual C++®
	`short GetSampleOnSynch();`
	`void SetSampleOnSynch(short nNewValue);`

Default 0 (OFF)	

SamplesPerMI	
Scope Vector	Description Sets the number of samples per burst interval.

Category
Standard
	Visual C++®
	`short GetSamplesPerMI();`
	`void SetSamplesPerMI(short nNewValue);`

Default 10	

SamplingMode	
Scope Vector	Description Select continuous sampling to collect data continuously, without pause. Select burst interval to collect data for a number of samples/sampling rate seconds. The burst interval is the time between each measurement burst. After each burst, the instrument will go to sleep to conserve power and recorder capacity. Values are 0 = Burst, 1 = Continuous.
Category Standard	

Default
1 (Continuous)
	Visual C++®
	`short GetSamplingMode();`
	`void SetSamplingMode(short nNewValue);`

SamplingRate	
Scope Vector/Vectrino	Description Sets the output rate for the velocity, amplitude, correlation, and pressure data. For Vector, the output rate for all other sensors (temperature, compass, tilt, etc.) is fixed to 1 second.
Category Standard	

Default	Visual C++®
	`short GetSamplingRate();`
	`void SetSamplingRate(short nNewValue);`

SamplingVolume	
Scope Vector/Vectrino	Description When reducing the sampling volume size, the total number of samples tused for the velocity calculation is reduced. The effect of this reduction is that the precision of the measured velocity is reduced. Values are 0 = 1.5 mm + TL, 1 = 4.6 mm + TL, 2 = 7.8 mm + TL, 3 = 10.9 mm + TL, 4 = 14 mm + TL. TL = TransmitLength.
Category Standard	

Default	Visual C++®
	`short GetSamplingVolume();`
	`void SetSamplingVolume(short newValue);`

SerialPort	
Scope All instruments	Description The name of the serial port connected to the instrument. Valid names are “COM1”, “COM2” etc.

Category
Communication
	Visual C++®
	`BSTR GetSerialPort();`
	`void SetSerialPort(LPCTSTR lpszNewValue);`

Default “COM1”	

SoundSpeed	
Scope All	Description Speed of sound to use if SondSpeedMode is Fixed.

Category
Standard (Vector/Vectrino)
Advanced (all others)
	Visual C++®
	`float GetSoundSpeed();`
	`void SetSoundSpeed(float newValue);`

Default 1525.0	

SoundSpeedMode	
Scope All	Description Speed of sound can be set by the user (Fixed) or calculated by the instrument based on the measured temperature and a user-input value for salinity (Measured). Values are 0 = Measured, 1 = Fixed.

Category
Standard (Vector/Vectrino)
Advanced (all others)
	Visual C++®
	`short GetSoundSpeedMode();`
	`void SetSoundSpeedMode(short nNewValue);`

Default 0 (Measured)	

StageAvgInterval	
Scope EasyQ	Description The stage averaging Interval specifies how long the instrument will be actively collecting stage data within each Measurement Interval. Example: Measurement Interval is set to 600 seconds, averaging interval set to 60 seconds means that the instrument will collect data for 1 minute every 10 minutes. The averaging Intervals (stage + flow) must be smaller than Measurement Interval.
Category Advanced	

Default
60
	Visual C++®
	`short GetStageAvgInterval();`
	`void SetStageAvgInterval(short nNewValue);`

StageMode	
Scope EasyQ	Description Enables or disables stage measurements. Values are 0 = OFF, 1 = ON.

Category
Standard
	Visual C++®
	`short GetStageMode();`
	`void SetStageMode(short nNewValue);`

Default 1 (ON)	

StageOffset	
Scope EasyQ	Description The StageOffset is added to the stage measurement after stage is computed. The stage algorithm first compares measured stage to pressure, and then adds the offset.
Category Standard	

Default
0
	Visual C++®
	`float GetStageOffset();`
	`void SetStageOffset(float newValue);`

StagePowerLevel	
Scope EasyQ	Description The stage power level sets how much acoustic energy the instrument transmits into the water for stage measurements. The difference between the highest level and the lowest level is about 20dB. Values are High = 0, HighLow = 1, LowHigh = 2, Low = 3.
Category Advanced	

Default
1 (HighLow)
	Visual C++®
	`short GetStagePowerLevel();`
	`void SetStagePowerLevel(short nNewValue);`

StartOnSynch	
Scope Vector/Vectrino	Description Enables or disables start data collection by external trigger signal.Values are 0 = OFF, 1 = ON.

Category
Advanced
	Visual C++®
	`short GetStartOnSynch();`
	`void SetStartOnSynch(short nNewValue);`

Default 0 (OFF)	

Timeout	
Scope All instruments	Description Instrument command response timeout in millisec.

Category
Communication
	Visual C++®
	`short GetTimeout();`
	`void SetTimeout(short nNewValue);`

Default 1000	

TransmitLength	
Scope Vector/Vectrino	Description The effect of increasing the transmit length is that the signal-to-noise ratio is increased. When changing the transmit length the available sampling volumes sizes is also changed. Values are 0 = 2 mm, 1 = 4 mm.

Category
Advanced (Vector)
Standard (Vectrino)
	Visual C++®
	`short GetTransmitLength();`
	`void SetTransmitLength(short nNewValue);`

Default 1 (4 mm)	

TriggerOut	
Scope Vector	Description Sets the output synch signal type for external devices. Values are 0 = Vector, 1 = Other

Category
Advanced
	Visual C++®
	`short GetTriggerOut();`
	`void SetTriggerOut(short nNewValue);`

Default 0 (Vector)	

VelRange	
Scope Vector/Vectrino	Description Nominal velocity range should be set to cover the range of the velocities anticipated during the deployment. A higher velocity range give more noise in the data and vice versa. See the User Guide for more information. Values are 0 = 7.0 m/s, 1 = 4.0 m/s, 2 = 2.0 m/s, 3 = 1.0 m/s, 4 = 0.3 m/s, 5 = 0.1 m/s, and 6 = 0.01 m/s.
Category Standard	

Default
4 (0.3 m/s)
	Visual C++®
	`short GetVelRange();`
	`void SetVelRange(short nNewValue);`

WaveASTThreshold	
Scope AWAC AST	Description This parameter is used in conjunction with Find First Peak, and specifies the minimum acoustic return to identify the free surface. This is a parameter has not units of measure, but 100 represents the default value for most cases.
Category Advanced	

Default
100
	Visual C++®
	`short GetWaveASTThreshold();`
	`void SetWaveASTThreshold(short nNewValue);`

WaveASTWindowSize	
Scope AWAC AST	Description This is the size of the AST window and should be designed such that the free surface is contained within it. Applies to AST systems only.
Category Advanced	

Default
10.0 m
	Visual C++®
	`short GetWaveASTWindowSize();`
	`void SetWaveASTWindowSize(short nNewValue);`

WaveBursts	
Scope Aquapro, AWAC	Description Enables or disables wave measurements. Values are 0 = OFF, 1 = ON.
Category Standard	

Default
1 (ON) if AWAC
0 (OFF) if Aquapro
	Visual C++®
	`short GetWaveBursts();`
	`void SetWaveBursts(short nNewValue);`

WaveCellPosition	
Scope AWAC (non AST)	Description This is the vertical position (distance from the instrument head) of the depth cell used for wave velocity measurements.
Category Advanced	

Default
15.0 m
	Visual C++®
	`float GetWaveCellPosition();`
	`void SetWaveCellPosition(float newValue);`

WaveCellSize	
Scope Aquapro, AWAC	Description The cell size specifies the vertical length in metres of the depth cell used for wave spectra velocity measurements.
Category Advanced	

Default	Visual C++®
	`float GetWaveCellSize();`
	`void SetWaveCellSize(float newValue);`

WaveFindPeak	
Scope AWAC AST	Description The standard method for finding the free surface is to use the maximum return from AST time series. Using this method chooses a first peak in the AST time series over a specified threshold (AST threshold). Applies to AST systems only.
Category Advanced	

Default
0 (Off)
	Visual C++®
	`short GetWaveFindPeak();`
	`void SetWaveFindPeak(short nNewValue);`

WaveInterval	
Scope Aquapro, AWAC	Description Sets how often the instrument will collect wave data (seconds).
Category Standard	

Default
3600
	Visual C++®
	`long GetWaveInterval();`
	`void SetWaveInterval(long nNewValue);`

WaveSamples	
Scope Aquapro, AWAC	Description Sets the number of data samples to collect for wave spectra computations. Values are 0 = 512, 1 = 1024, and 2 = 2048.

Category
Standard
	Visual C++®
	`short GetWaveSamples();`
	`void SetWaveSamples(short nNewValue);`

Default 1 (1024)	

WaveSamplingRate	
Scope Aquapro, AWAC	Description Sets the sampling rate to use for wave data collection. Values are 0 = 1Hz, 1 = 2Hz.

Category
Standard
	Visual C++®
	`short GetWaveSamplingRate();`
	`void SetWaveSamplingRate(short nNewValue);`

Default 0 (1 Hz)	

WaveStaticMode	
Scope AWAC	Description This mode is chosen over the automatic mode when the user wants to have a fixed positioned velocity cell position or AST window. One example deployment would be at a location with no tidal variations and wave heights that are small. Generally speaking this mode is rarely used.

Category
Advanced
	Visual C++®
	`short GetWaveStaticMode();`
	`void SetWaveStaticMode(short nNewValue);`

Default 0 (Off)	

WaveASTWindowStart (AST systems only)	
Scope AWAC AST	Description This is where the AST window begins to measure, referenced from the centre transducer in the direction of the centre beam (beam 4). Applies to AST systems only.

Category
Advanced
	Visual C++®
	`short GetWaveASTWindowStart();`
	`void SetWaveASTWindowStart(short nNewValue);`

Default 15.0 m	

Methods

The following methods are used to control the instrument data collection.

Connect	
Scope	Description
All instruments	Connects to the instrument and checks its status. Returns TRUE if successful. **Call this method prior to any other method!**
	Visual C++® `BOOL Connect();`

DialModem	
Scope	Description
All instruments	Dials the phone number. Returns TRUE if successful.
	Visual C++® `BOOL DialModem();`

Disconnect	
Scope	Description
All instruments	Disconnects from the instrument. Returns TRUE if successful.
	Visual C++® `BOOL Disconnect();`

DownloadData	
Scope	Description
All instruments, except Vectrino	Downloads (dumps) data from the recorder. Returns recorder file stop address (end of file) strFileName - name of disk file nStartAddress - recorder address to start from (most recent data) 0 = read all data
	Visual C++® `long DownloadData(LPCTSTR strFileName, long nStartAddress, BOOL bAppend)`

EraseRecorder	
Scope	Description
All instruments, except Vector/Vectrino	Erases all recorder files
	Visual C++® `BOOL EraseRecorder()`

GetAmp	
Scope	Description
All instruments	Gets the most recent amplitude data for the specified cell and beam (counts). Cell and beam numbering start at 1.
	Visual C++® `short GetAmp(short nCell, short nBeam)`

GetAmplitude	
Scope	Description
All instruments	Gets the most recent amplitude data for the specified cell number. Returns TRUE if successful.
	Visual C++® `BOOL GetAmplitude(short nCell, short FAR* nAmp1, short FAR* nAmp2, short nAmp2, short FAR* nAmp3);`

GetAnalogInput	
Scope	Description
All instruments, except Vectrino	Gets the most recent analogue input data for the specified channel (counts). Channel is 1 or 2.
	Visual C++® `short GetAnalogInput(short nChannel)`

GetAnalogInputs	
Scope	Description
All instruments, except Vectrino	Gets the most recent analog input data. Returns TRUE if successful.
	Visual C++® `BOOL GetAnalogInputs(short FAR* nVal1, short FAR* nVal2);`

GetBattery	
Scope	Description
All instruments	Gets the most recent battery voltage (V).
	Visual C++® `float GetBattery()`

GetClock	
Scope	Description
All instruments	Returns the current date and time of the RTC in the instrument. Returns 0 if not successful.
	Visual C++® `DATE GetClock();`

GetConfig	
Scope	Description
All instruments	Reads the configuration (properties) from the instrument. Returns TRUE if successful.
	Visual C++® `BOOL GetConfig();`

GetDateTime	
Scope	Description
All instruments	Get date and time for the most recent data.
	Visual C++® `DATE GetDateTime();`

GetError	
Scope	Description
All instruments	Gets the most recent error word (V).
	Visual C++® `short GetErrorWord()`

GetFileName	
Scope	Description
All instruments, except Vectrino	Returns the recorder filename
	Visual C++® `BSTR GetFileName(short nIndex)`

GetFirmwareVersion	
Scope	Description
All instruments	Returns the firmware version. Must be called after Connect() which reads the hardware and software info from the instrument.
	Visual C++® `BSTR GetFirmwareVersion();`

GetFirstFile	
Scope	Description
All instruments, except Vectrino	Finds the first data file in the recorder and returns the file position number to be used in consequtive calls to GetNextFile. Returns –1 if there are no files on the recorder.
	Visual C++® `short GetFirstFile(BSTR FAR* strFileName);`

GetHeading	
Scope	Description
All instruments, except Vectrino	Gets the most recent compass heading (°).
	Visual C++® `float GetHeading()`

GetId	
Scope	Description
All instruments	Returns the instrument ID string.
	Visual C++® `BSTR GetID()`

GetLastError	
Scope	Description
All instruments	Gets the most recent error message string.
	Visual C++® `BSTR GetLastError()`

GetLastMessage	
Scope	Description
All instruments	Returns the last error message as a string. If the Messages property is ON all errors are reported by the PdCommX control. If Messages is OFF, the application can use GetLastMessage to control the message pop-up itself.
	Visual C++® `BSTR GetLastMessage()`

GetModemReply	
Scope	Description
All instruments	Returns the modem response string.
	Visual C++® `BSTR GetModemReply()`

GetNextFile	
Scope	Description
All instruments, except Vectrino	Finds the next data file in the recorder and returns the file position number to be used in consequtive calls to GetNextFile.
	Visual C++® `short GetNextFile(short nPos, BSTR FAR* strFileName);`

GetNoiseAmp	
Scope	Description
All instruments	Gets the noise amplitude for the specified beam (counts). Beam numbering starts at 1.
	Visual C++® `short GetNoiseAmp(short nBeam)`

GetNoiseLevel	
Scope	Description
All instruments	Gets the instrument noise level measurement (counts).
	Visual C++® `boolean GetNoiseLevel(short* nAmp1, short* nAmp2, short* nAmp3, short* nAmp4)`

GetNumFiles	
Scope	Description
All instruments; except Vectrino	Returns the number of files in the instrument recorder.
	Visual C++® `short GetNumFiles()`

GetPitch	
Scope All instruments, except Vectrino	Description Gets the most recent pitch measurement (°).
	Visual C++® `float GetPitch()`

GetPressure	
Scope All instruments, except Vectrino	Description Gets the most recent pressure data.
	Visual C++® `float GetPressure();`

GetRangeNBeams	
Scope EasyQ	Description Gets the number of beams used for range measurement.
	Visual C++® `short GetRangeNBeams()`

GetRoll	
Scope All instruments, except Vectrino	Description Gets the most recent roll measurement (°).
	Visual C++® `float GetRoll()`

GetSensors	
Scope All instruments, except Vector	Description Gets the most recent sensors data. Returns TRUE if successful.
	Visual C++® `BOOL GetSensors(float FAR* fTemperature, float FAR* fPitch, float FAR* fRoll, float FAR* fHeading);`

GetSerialNo	
Scope All instruments	Description Returns the instrument ID string.
	Visual C++® `BSTR GetSerialNo()`

GetSNR	
Scope	Description
All instruments	Gets the most recent SNR data for the specified cell number. Returns TRUE if successful.
	Visual C++® `BOOL GetSNR(short nCell, float FAR* fB1, float FAR* fB2, float FAR* fB3);`

GetSSpeed	
Scope	Description
All instruments	Gets the most recent sound speed used (m/s).
	Visual C++® `float GetSSpeed()`

GetStage	
Scope	Description
EasyQ	Gets the most recent stage measurement (m).
	Visual C++® `float GetStage()`

GetStageAndQuality	
Scope	Description
EasyQ	Gets the most recent stage (m) and quality measurements.
	Visual C++® `boolean GetStageAndQuality(float* fStage, short* nQuality, float* fStageP, short* nQualityP)`

GetStageBlanking	
Scope	Description
EasyQ	Gets the most recent stage blanking distance (m) (EasyQ).
	Visual C++® `float GetStageBlanking()`

GetStageQualityP	
Scope EasyQ	Description Gets the most recent quality number.
	Visual C++® `short GetStageQualityP()`

GetStatus	
Scope All instruments	Description Gets the most recent status data. Returns TRUE if successful.
	Visual C++® `BOOL GetStatus(float FAR* fSoundSpeed, float FAR* fBattery, short FAR* nError, short FAR* nStatus);`

GetStatusWord	
Scope All instruments	Description Gets the most recent status word.
	Visual C++® `short GetStatusWord()`

GetTemperature	
Scope All instruments	Description Gets the most recent temperature measurement (°C).
	Visual C++® `float GetTemperature()`

GetVarAmplitude	
Scope All instruments	Description Gets the most recent amplitude data as a variant type array.
	Visual C++® `VARIANT GetVarAmplitude();`

GetVarAnalogInputs	
Scope All instruments, except Vectrino	Description Gets the most recent analogue input data as a variant type array.
	Visual C++® `VARIANT GetVarAnalogInputs();`

GetVarNoiseLevel	
Scope	Description
All instruments	Gets the instrument noise level measurement as a variant type array (counts).
	Visual C++® `VARIANT GetVarNoiseLevel()`

GetVarSensors	
Scope	Description
All instruments, except Vectrino	Gets the most recent sensors data as a variant type array.
	Visual C++® `VARIANT GetVarSensors();`

GetVarSNR	
Scope	Description
All instruments	Gets the most recent SNR data as a variant type array.
	Visual C++® `VARIANT GetVarSNR();`

GetVarStage	
Scope	Description
EasyQ	Gets the most recent stage measurement (m) as a variant type array.
	Visual C++® `VARIANT GetVarStage()`

GetVarStatus	
Scope	Description
All instruments, except Vectrino	Gets the most recent status data as a variant type array.
	Visual C++® `VARIANT GetVarStatus();`

GetVarVelocity	
Scope	Description
All instruments	Gets the most recent velocity data as a variant type array.
	Visual C++® `VARIANT GetVarVelocity();`

GetVarWaveAmplitude	
Scope	Description
AWAC, Aquapro	Gets the most recent wave burst amplitude data as a variant type array (counts).
	Visual C++® `VARIANT GetVarWaveAmplitude()`

GetVarWaveVelocity	
Scope	Description
AWAC, Aquapro	Gets the most recent wave burst velocity data (m/s) as a variant type array.
	Visual C++® `VARIANT GetVarWaveVelocity()`

GetVel	
Scope	Description
All instruments	Gets the most recent velocity data for the specified cell and beam (m/s). Cell and beam numbering start at 1
	Visual C++® `float GetVel(short nCell, short nBeam)`

GetVelocity	
Scope	Description
All instruments	Gets the most recent velocity data for the specified cell number. Returns TRUE if successful.
	Visual C++® `BOOL GetVelocity(short nCell, float FAR* fVel1XE, float FAR* fVel2YN, float FAR* fVel3ZU);`

GetWaveAmp	
Scope	Description
AWAC, Aquapro	Gets the most recent wave burst amplitude data (counts) for the specified beam. Beam numbering starts at 1.
	Visual C++® `short GetWaveAmp(short nBeam)`

GetWaveAmplitude	
Scope AWAC, Aquapro	Description Gets the most recent wave burst amplitude data (counts). Returns TRUE if successful.
	Visual C++® `BOOL GetWaveAmplitude(short FAR* nAmp1, short FAR* nAmp2, short FAR* nAmp3, short FAR* nAmp4)`

GetWaveDateTime	
Scope AWAC, Aquapro	Description Gets Date and Time for the most recent wave burst measurement.
	Visual C++® `DATE GetWaveDateTime()`

GetWaveDistance	
Scope AWAC with AST	Description Gets the most recent wave AST distance measurement (m).
	Visual C++® `float GetWaveDistance()`

GetWaveDistance2	
Scope AWAC with AST	Description Gets the most recent wave AST distance2 measurement (m).
	Visual C++® `float GetWaveDistance2()`

GetWavePressure	
Scope AWAC, Aquapro	Description Gets the most recent wave burst pressure data (m).
	Visual C++® `float GetWavePressure()`

GetWaveQuality	
Scope AWAC with AST	Description Gets the most recent wave AST quality number (counts).
	Visual C++® `short GetWaveQuality()`

GetWaveVel	
Scope AWAC, Aquapro	Description Gets the most recent wave burst velocity data (m/s) for the specified beam. Beam numbering starts at 1.
	Visual C++® `float GetWaveVel(short nBeam)`

GetWaveVelocity	
Scope AWAC, Aquapro	Description Gets the most recent wave burst velocity data (m/s). Returns TRUE if successful.
	Visual C++® `BOOL GetWaveVelocity(float FAR* fVel1, float FAR* fVel2, float FAR* fVel3, float FAR* fVel4)`

HangUpModem	
Scope All instruments	Description Hangs up the modem. Returns TRUE if successful.
	Visual C++® `BOOL HangUpModem();`

InitializeModem	
Scope All instruments	Description Initializes the modem. Returns TRUE if successful.
	Visual C++® `BOOL InitializeModem();`

InquireState	
Scope	Description
All instruments	Returns the instrument state. Values are 0 = Firmware upgrade mode, 1 = Measurement mode, 2 = Command mode, 4 = Data retrieval mode, 5 = Confirmation mode. Returns –1 if unknown or not successful.
	Visual C++® `short InquireState();`

IsConnected	
Scope	Description
All instruments	Returns TRUE if connected.
	Visual C++® `BOOL IsConnected();`

IsModemOnline	
Scope	Description
All instruments	Returns TRUE if the modem is online.
	Visual C++® `BOOL IsModemOnline()`

LoadDeployment	
Scope	Description
All instruments, except Vectrino	Loads the specified deployment (.dep) file and configures the instrument. Deployment files are instrument configuration files generated and saved uring the standard instrument software.
	Visual C++® `boolean LoadDeployment(BSTR strFileName)`

ReadFile	
Scope	Description
All instruments, except Vectrino	Downloads a data file to the specified disk filename. Performs a CRC check on the data if bCRC = TRUE. Returns TRUE if successful.
	Visual C++® `BOOL ReadFile(short nPos, LPCTSTR strDiskFileName, BOOL bCRC);`

SendModemCommand	
Scope	Description
All instruments	Sends a modem command string. Returns TRUE if successful. Use **GetModemReply** to get the modem response.
	Visual C++® `BOOL SendModemCommand(LPCTSTR strCommand)`

SetClock	
Scope	Description
All instruments	Sets the date and time of the RTC in the instrument. Returns TRUE if successful.
	Visual C++® `BOOL SetClock(DATE dateTime);`

SetConfig	
Scope	Description
All instruments	Writes the configuration (properties) to the instrument. Returns TRUE if successful. Note that for most of the property settings to take effect, the **SetConfig** method must be called prior to Start command. See also **Connect**.
	Visual C++® `BOOL SetConfig();`

SetInstrumentBaudRate	
Scope	Description
All instruments	Sets the instrument baudrate. Set bSave = TRUE to make permanent.
	Visual C++® `BOOL SetInstrumentBaudRate(long nBaudRate, BOOL bSave)`

Start	
Scope	Description
All instruments	Starts online measurements based on the current configuration of the instrument. If bRecorder = TRUE data is stored to a new file in the recorder. Returns TRUE if successful. See also **SetConfig** (above).
	Visual C++® `BOOL Start(BOOL bRecorder);`

StartDeployment	
Scope	Description
All instruments, except Vectrino	Starts a deployment based on the current configuration of the instrument. Data is stored to a new file in the recorder. Returns TRUE if successful.
	Visual C++® `BOOL StartDeployment();`

StartDiskRecording	
Scope	Description
All instruments	Starts data recording to disk. Specify the filename without extension. If AutoName = TRUE a new file will be opened for data recording each time the specified time interval has elapsed. The current date and time is then automatically added to the filename. Returns TRUE if successful.
	Visual C++® `BOOL StartDiskRecording(CString strFileName, BOOL bAutoName);`

Stop	
Scope	Description
All instruments	Stops online measurements and deployment data recording. Returns TRUE if successful.
	Visual C++® `BOOL Stop();`

StopDiskRecording	
Scope	Description
All instruments	Stops data recording to disk.
	Visual C++® `BOOL StopDiskRecording();`

Events

The following event is used to notify the client application that new data is available.

OnNewData

Scope

All instruments

Description

Notifies that new data is available from the instrument. Use the Get... methods to read the data. The Type parameter specifies the data type. Data type values are:

01	Measurement data (Aquadopp)
02	Distance data (Vectrino)
80	Diagnostics data (Aquadopp)
06	Diagnostics header data (Aquadopp)
07	Probe check data (Aquadopp/Vector)
08 or 09 or 10	Velocity data (Vector)
11	Sensor data (Vector)
12	Velocity header data (Vector)
20	Profile data (AWAC)
21 or 22 or 23	Profile data (Aquapro)
24 or 25 or 26	Profile data (Continental)
28	Profile and distance data (Aquapro)
30	Wave data (AWAC and Aquapro)
31	Wave header data (AWAC & Aquapro)
40	Measurement data (EasyQ)
41	Range check data (EasyQ)
42	Stage check data (EasyQ)
50	Velocity header data (Vectrino)
51	Velocity data (Vectrino)

Visual C++®

```
void OnNewData(short hType);
```

CHAPTER 4
Implementation Examples

Implementation examples are provided in Visual C++®, Visual Basic® and Delphi™.

Visual C++®

```
BOOL COCXTestDlg::OnInitDialog()
{
   CDialog::OnInitDialog();
   .
   .
   .

   m_pdcommx.SetMeasInterval(1);
   m_pdcommx.SetAvgInterval(1);
   m_pdcommx.SetDiagnosticsMode(0) ;

   return TRUE;
}

void COCXTestDlg::OnBtnStartstop()
{
   CString str;

   m_btnStartStop.GetWindowText(str);
   if (str == "&Start") {
```

```
      m_pdcommx.Connect();
      if (m_pdcommx.InquireState() != 2) {   // not in command mode
         m_pdcommx.Stop();
         m_pdcommx.Disconnect();
         return;
      }
      if (!m_pdcommx.SetConfig()) {
         m_pdcommx.Stop();
         m_pdcommx.Disconnect();
         return;
      }
      if (!m_pdcommx.Start(FALSE)) {
         m_pdcommx.Stop();
         m_pdcommx.Disconnect();
         return;
      }
      m_btnStartStop.SetWindowText("&Stop");
   }
   else {
      m_pdcommx.Stop();
      m_pdcommx.Disconnect();
      m_btnStartStop.SetWindowText("&Start");
   }
}

BEGIN_EVENTSINK_MAP(COCXTestDlg, CDialog)
    //{{AFX_EVENTSINK_MAP(COCXTestDlg)
   ON_EVENT(COCXTestDlg, IDC_PDCOMMXCTRL, 1 /* OnNewData */, OnOnNewDataPdcommxctrl,
VTS_I2)
   //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void COCXTestDlg::OnOnNewDataPdcommxctrl(short hType)
{
   float dVel[3];
   float dTemperature, dPitch, dRoll, dHeading, dPressure;
   COleDateTime odtTime;
   short nAmp[3];

   odtTime = m_pdcommx.GetDateTime();
   m_pdcommx.GetVelocity(0, &dVel[0], &dVel[1], &dVel[2]);
   m_pdcommx.GetAmplitude(0, &nAmp[0], &nAmp[1], &nAmp[2]);
   dPressure = m_pdcommx.GetPressure();
   m_pdcommx.GetSensors(&dTemperature, &dPitch, &dRoll, &dHeading);

   m_strVel1.Format("%.2f",dVel[0]);
   m_strVel2.Format("%.2f",dVel[1]);
   m_strVel3.Format("%.2f",dVel[2]);
   m_strAmp1.Format("%d",nAmp[0]);
```

```
    m_strAmp2.Format("%d",nAmp[1]);
    m_strAmp3.Format("%d",nAmp[2]);

    m_strTemp.Format("%.1f",dTemperature);
    m_strPitch.Format("%.1f",dPitch);
    m_strRoll.Format("%.1f",dRoll);
    m_strHeading.Format("%.1f",dHeading);

    m_strPressure.Format("%.1f",dPressure);
    m_strTime = odtTime.Format();

    UpdateData(FALSE);
}
```

Visual Basic®

```
Dim DateTime As Date
Dim Vel(3) As Single
Dim Amp(3) As Integer
Dim Temperature As Single
Dim Pitch As Single
Dim Roll As Single
Dim Heading As Single
Dim Pressure As Single

Private Sub cmdStartStop_Click()
If InStr(cmdStartStop.Caption, "Start") > 0 Then
    pdcommx.Connect
    If pdcommx.InquireState <> 2 Then  'not in command mode
        pdcommx.Stop
        pdcommx.Disconnect
        Exit Sub
    End If
    'initialize instrument
    If Not pdcommx.SetConfig Then
        pdcommx.Stop
        pdcommx.Disconnect
        Exit Sub
    End If
    If Not pdcommx.Start(False) Then
        pdcommx.Stop
        pdcommx.Disconnect
        Exit Sub
    End If
    cmdStartStop.Caption = "&Stop"
```

```
Else
    pdcommx.Stop
    pdcommx.Disconnect
    cmdStartStop.Caption = "Start"
End If
End Sub

Private Sub Form_Load()
    pdcommx.MeasInterval = 1
    pdcommx.AvgInterval = 1
    pdcommx.DiagnosticsMode = DiagOff
    ' more properties .... or use property window
End Sub

Private Sub pdcommx_OnNewData(ByVal hType As Integer)
    DateTime = pdcommx.GetDateTime
    Call pdcommx.GetVelocity(0, Vel(1), Vel(2), Vel(3))
    Call pdcommx.GetAmplitude(0, Amp(1), Amp(2), Amp(3))
    Pressure = pdcommx.GetPressure
    Call pdcommx.GetSensors(Temperature, Pitch, Roll, Heading)

    lblVel1.Caption = Str(Vel(1))
    lblVel2.Caption = Str(Vel(2))
    lblVel3.Caption = Str(Vel(3))
    lblAmp1.Caption = Str(Amp(1))
    lblAmp2.Caption = Str(Amp(2))
    lblAmp3.Caption = Str(Amp(3))

    lblTemp.Caption = Str(Temperature)
    lblPitch.Caption = Str(Pitch)
    lblRoll.Caption = Str(Roll)
    lblHead.Caption = Str(Heading)

    lblPress.Caption = Str(Pressure)
    lblTime.Caption = Str(DateTime)
End Sub
```

Delphi™

```
var
  OCXTestForm: TOCXTestForm;
  DateTime: TDateTime;
  Vel: array[1..3] of Single;
  Amp: array[1..3] of Smallint;
  Temperature, Pitch, Roll, Heading, Pressure: Single;
```

```
  I, J, K: Smallint;

procedure TOCXTestForm.FormCreate(Sender: TObject);
begin
  PdCommX.MeasInterval := 1;
  PdCommX.AvgInterval := 1;
  PdCommX.DiagnosticsMode := 0;
  // more properties .... or use property window
end;

procedure TOCXTestForm.btnStartStopClick(Sender: TObject);
begin
  if btnStartStop.Caption = '&Start' then
    begin
      PdCommX.Connect;
      if PdCommX.InquireState <> 2 then  // not in command mode
        begin
          PdCommX.Stop;
          PdCommX.Disconnect;
          Exit;
        end;

      // initialize instrument
      if PdCommX.SetConfig <> True then
        begin
          PdCommX.Stop;
          PdCommX.Disconnect;
          Exit;
        end;

      if PdCommX.Start(0) = 0 then
        begin
          PdCommX.Stop;
          PdCOmmX.Disconnect;
          Exit;
        end;
      btnStartStop.Caption := '&Stop'
    end
  else
    begin
      PdCommX.Stop;
      PdCommX.Disconnect;
      btnStartStop.Caption := '&Start';
    end;
end;

procedure TOCXTestForm.NewData(Sender: TObject; hType: Smallint);
begin
  DateTime := PdCommX.GetDateTime;
```

```
  PdCommX.GetVelocity(0, Vel[1], Vel[2], Vel[3]);
  PdCommX.GetAmplitude(0, Amp[1], Amp[2], Amp[3]);
  Pressure := PdCommX.GetPressure;
  PdCommX.GetSensors(Temperature, Pitch, Roll, Heading);

  edtVel1.Text := FloatToStrF(Vel[1],ffFixed,7,2);
  edtVel2.Text := FloatToStrF(Vel[2],ffFixed,7,2);
  edtVel3.Text := FloatToStrF(Vel[3],ffFixed,7,2);
  edtAmp1.Text := IntToStr(Amp[1]);
  edtAmp2.Text := IntToStr(Amp[2]);
  edtAmp3.Text := IntToStr(Amp[3]);

  edtTemp.Text := FloatToStrF(Temperature,ffFixed,7,1);
  edtPitch.Text := FloatToStrF(Pitch,ffFixed,7,1);
  edtRoll.Text := FloatToStrF(Roll,ffFixed,7,1);
  edtHead.Text := FloatToStrF(Heading,ffFixed,7,1);

  edtPress.Text := FloatToStrF(Pressure,ffFixed,7,1);
  edtTime.Text := DateTimeToStr(DateTime);
end;

end.
```

Nortek AS, Vangkroken 2, NO-1351 Rud, Norway. Tel +47 67 17 45 00. Fax +47 67 13 67 70. inquiry@nortek.no. www.nortek-as.com