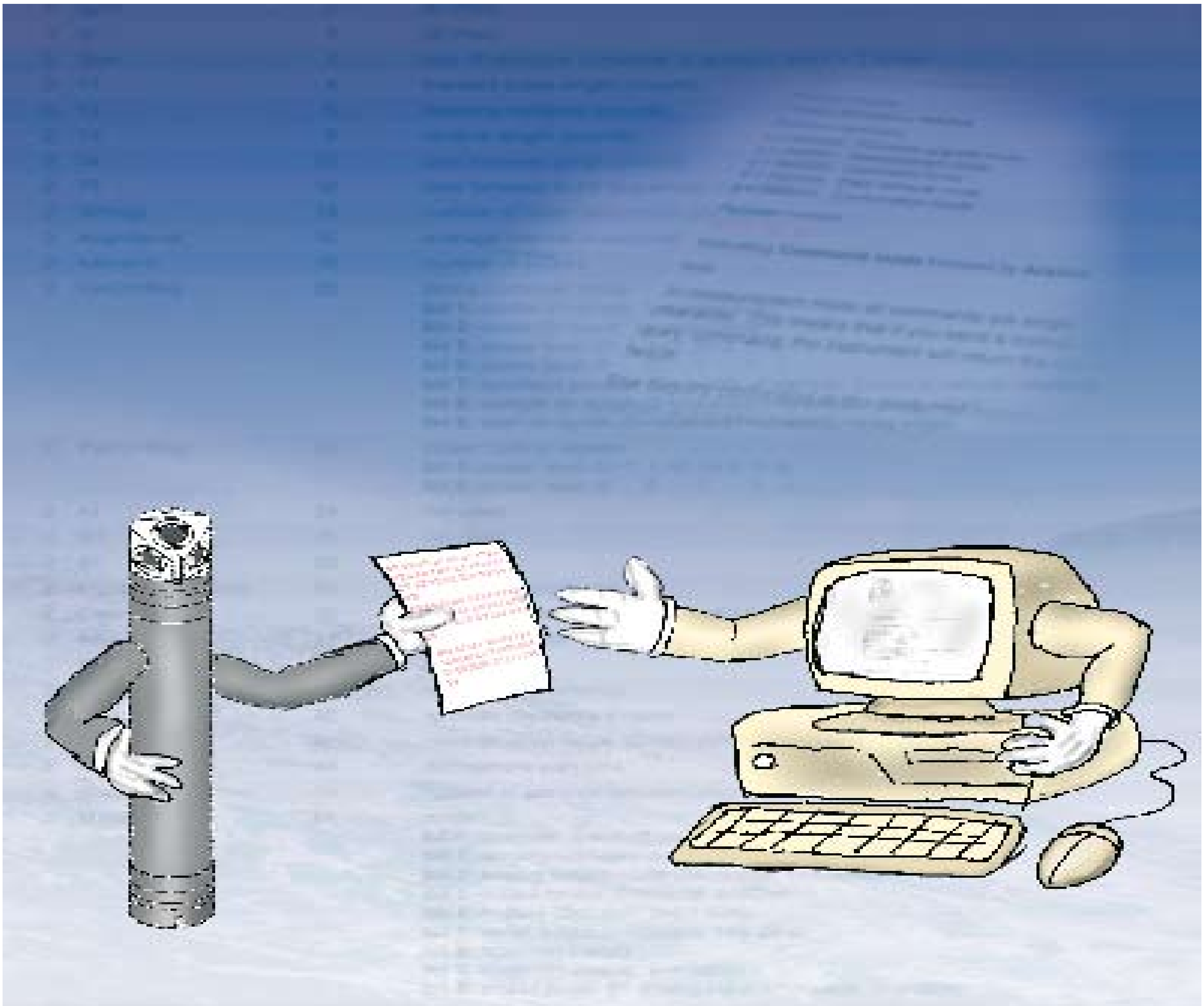
NORTEK AS

SYSTEM
INTEGRATOR GUIDE



PARADOPP FAMILY OF PRODUCTS

Febuary 2010

SYSTEM INTEGRATOR GUIDE

PARADOPP FAMILY OF PRODUCTS

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

Tel: +47 6717 4500 • Fax: +47 6713 6770 • e-mail: inquiry@nortek.no • www.nortek-as.com

Contents

Chapter 1
Introduction

This document provides the information needed to control a Nortek Paradopp product (Aquadopp, Vector, etc.) with a non-PC controller. It is aimed at system integrators and engineers with interfacing experience. Code examples are provided in C. The document's scope is limited to interfacing and does not address general performance issues of the instruments. For a more thorough understanding of the principle of operation, we recommend the user guide that accompanies the individual instruments.

The document is complete in the sense that it describes all available commands and modes of communication. For most users, it will make sense to let the supplied Nortek software do most of the hardware configuration and then let the controller limit its task to starting/stopping data collection. For more in-depth information about specific commands, we urge you to contact Nortek to discuss how your particular problem is best solved. For those who wish to who write their own Windows applications to control one or more Paradopp products an ActiveX® object is available. This greatly simplifies interfacing and the handling of the internal data structures.

Note that the Paradopp products use a binary data format for communication. This makes it hard to "see" what is going on with a terminal emulator. However, the binary interface saves programming time because parsing the text files won't be needed. It may take more time initially to put the basic communication in place, but once done the remainder of the work should be straightforward. The use of checksums and CRC helps to make the binary data interface more robust.

As always, these types of documents are subject to change. We recommend that you contact Nortek to ensure you have the all the latest information and versions of any software you plan to use. We recommend you do this as part of your project planning before you start any development work. If you have any comments or suggestions on the information given here, please let us know. Your comments are always appreciated, our general e-mail address is inquiry@nortek.no. You can always join our forum and post your comments, suggestions or questions there, visit our website www.nortek-as.com and click the link to the forum.

ActiveX®

The ActiveX®/DLL software interface provides functions to configure the instrument, control the data acquisition process and retrieve data from the recorder. In a DLL implementation C/C++ API calls are made to the Paradopp DLL. A Paradopp OCX implementation requires that the software development environment supports the OCX interface. Visual Basic, Visual C++ and Delphi, are a few environments that support the OCX interface.

The ActiveX® control interface is described in the ActiveX Module for System Integrators, available separately from Nortek.

Chapter 2
Basic Interface Concepts

The Paradopp family of products communicates with a default protocol of 8 data bits, no parity and 1 stop bit. The baud rate is user selectable and can be configured either with the supplied Windows programs or by using direct commands to the system after the direct communication has been initiated (see the chapter on Remote Control Terminal Commands).
The only lines used are RxD, TxD, and GND. Status and handshaking lines are not used.

The Break Command

A break command is used to change between the various operational modes of the instrument and to interrupt the instrument regardless of which mode it is in. It is used frequently when communicating with the instrument. Consequently, any system designed to control a Paradopp system must be able to send a break.

The operational modes for any Paradopp system are:

- Command mode. The system is waiting for an instruction over the serial line. After 5 minutes of inactivity, the system will power down.
- Power down mode. This state is used to conserve power. A break must be sent to cause the instrument to wake up.
- Measurement mode. The system cycles through a series of states when collecting data. To exit collection mode, a break and confirmation string must be sent.
- Data retrieval mode.

After a power on/off, the system will remember what mode it is in.

Tip: When you send a break to the instrument, it will respond as follows (using Aquadopp Profiler as an example):

AQUAPRO
NORTEK 2003
Version 1.23
Command mode

Determining Which Type of Break to Use

There are two types of break commands in Nortek instruments, a soft break and a hard break.

Traditionally, sending a break used to be done by holding the transmit line high on the serial line for period of 500 ms. This is referred to as hard break. If you have direct connection between the controlling device and the instrument this will work fine. However, if other devices are inserted in the communication path, problems may arise. Certain GSM modems, for example, handle breaks in their own way. Typically, they have no problems in accepting a 500 ms break on the input, but they only output a 100 ms break. This may not be recognised as a break by the instrument.

It may happen that a break is sent to the instrument at exactly the same time as the instrument receives an interrupt from its Real Time Clock (RTC). In such cases the RTC is always given priority.

This implies that if you receive no acknowledgement of the break you sent, you will need to send another.

Hence, your controller should be set up to verify that the break was actually received by the instrument before you proceed.

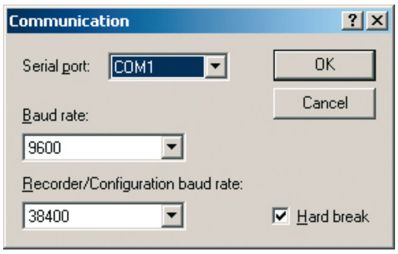
To get around this problem we introduced the soft break, which consists entirely of characters. It can be used without problems with any device capable of RS 232 or RS 422 communication.

Which break command to use for a particular instrument depends on the production date of the instrument, and most instruments manufactured after 2001 use soft break.

The easiest way to find out which type of break command to use is to run the Nortek software shipped with the instrument.

To check which type of break command to use:

1. Run the deployment setup software shipped with the instrument and initiate an auto detect by clicking Deployment > Planning > Load from instrument.
2. Click Communication > Serial Port to produce a dialogue box like the one shown below.



3. When the Hard break box is checked a hard break is to be used. If unchecked a soft break is to be used.

Hexadecimal values are prefixed by 0x

Example: 0x4a2b

Checksum Control

Most data structures contain a 16-bit checksum. An example program is given in the chapter on Data Structures to help explain how the checksum can be implemented.

Two-character ASCII Commands

The command interface uses two character commands where the two characters are treated as a single 16-bit word. The time delay between the two characters in a command must be less than 0.5 second, otherwise both characters will be discarded by the Paradopp. Data is transferred as words and the convention is Intel style, which means that low byte is sent before high byte. The data types are given in the section describing the various commands. More about this can be found in the Terminal Commands chapter.

Acknowledgement

After a successful command is sent, the system returns an acknowledgement. The actual value for acknowledge (AckAck) is 0x0606. Whenever the system firmware receives a command/word that is invalid, it immediately returns a negative acknowledge (NackNack). The value is 0x1515.

Chapter 3
Use with a Controller

Always use the accompanying Nortek software to set up the instrument for deployment.

This chapter provides useful information when setting up your Nortek instrument with a controller. However, we recommend that you read through the user guide that came with your Nortek instrument before you embark on your controller project.

Basically, a controller will act in one of the two following ways:
- As a simple storage unit for the data acquired.
- As a device controlling the Nortek instrument's behaviour, with or without data transfer to the controller.

All Nortek instruments come with deployment software running on the Windows® platform. We strongly recommend that you use this software to set up the instrument properly.

The data output to the controller is in binary format for all instruments. However, the Aquadopp Profiler, the Aquadopp Current Meter and the Continental can output data in ASCII format – see the chapter on ASCII Output for more information.

All Nortek instruments are supplied with RS 232 interface unless specified otherwise. For long distance transmission (more than 50–100 m) we recommend the use of RS 422, which is available as an option for all Paradopp instruments.

Simple Storage Device

If you decide to use your controller as a simple storage device you will have to make up your mind whether or not to use the internal recorder in addition to the controller. More about the internal recorder feature can be found in the user guide for the Nortek instrument.

Data output from the Nortek instrument will be properly time stamped as long as the instrument remains powered, so you won't have to implement time stamping in the controller to keep track of the data acquisition.

See the chapter on Data Structures for more information on how to interpret the data received from the instrument.

Control the Instrument directly

If you choose to download deployment setups from the controller during deployment, you may want to send a GA command every time you download a new setup to store all configuration data together with the corresponding measurement data.

Deployment setups should always be generated by means of the software accompanying your Nortek instrument.

If you decide to use your controller to control the Nortek instrument, you have two options:

- The controller starts and stops the measurements by turning the power to the Nortek instrument off when not measuring. This allows for a longer deployment. However, this may require that the controller to time-stamp the data, since Nortek instruments may loose their time information when the power is removed for more than 5 minutes. For more information regarding power consumption read the power consumption paragraph below.
- The controller starts and stops the measurements using a combination of a two character ASCII command and a break command.

You may want to store data read from the instrument in the controller. Some applications may also require that you download deployment setups from the instrument at regular intervals and store these in the controller.

For commands to be received and executed, the instrument must be in Command mode. If the instrument is in Power down mode a break must be sent to wake it up. If, on the other hand, the instrument is in Data collection mode (i.e. measuring) a break followed by a confirmation string must be sent. The confirmation string will be the MC command, which must be sent within 10 seconds after the break. Otherwise, the instrument will resume data measurement. In some newer versions of the firmware the confirmation string - MC command - must be sent within 60 seconds or the instrument will resume data measurement.

Turning the Power On/Off

When using this technique of removing and applying power, the recorder will operate in Append mode. If started with the SR command (Start with Recorder), so that all the data will be recorded to the same file. This opens up for easy verification of the controller since what has been logged to the internal recorder is identical to what has been sent to the controller.

In this case, the system will automatically start measuring and outputting data when power is applied. To use this method effectively, you must:

- Make sure the appropriate system configuration has been downloaded to the instrument, either from a PC or from the controller.
- Start data collection from the PC or the controller before disconnecting. Once the power is shut down, the instrument will remember that it is in data collection mode and continue to collect data once the power is re-applied.

Power consumption

On older instruments or those with firmware versions prior to V3.0 the internal clock may lose the correct time if the power is disconnected for more than 5 minutes. Instruments with later versions of the firmware will maintain the correct time for several weeks. Newer versions of the instruments use so little power that you will need to disconnect the power for some time or configure the instrument for continuous operation. If not the power loss might not be detected by the instrument.

Over the Serial Line

In this case, the data collection is controlled over the serial line. To start data collection, the two-character ASCII command AD (AquireData) is sent. The instrument automatically enters power down when the measurement is finished. To wake the system up or interrupt the measurement – a break must be sent. Using the AD command the instrument will jump directly to command mode upon receiving a break.

To start a measurement from Command mode, send the command ST. The system will send an acknowledge (AckAck) to show that the measurement is started. More about this can be found in the Terminal Commands chapter.

Always use the software accompanying your Nortek instrument to generate deployment files. The Nortek software must be set in a special mode to generate binary format deployment files – see CC command for details.

A typical sequence proceeds as follows:

- Send a break command to gain control of the system and put it in Command mode. If the system is busy collecting data (i.e. measuring), a verification is required, otherwise the instrument will not stop measuring. Send the characters MC within 60 seconds.
- To start a measurement from Command mode send the command ST. (SR if you want to also store data in the instrument's recorder – see the list of commands in Terminal Commands chapter.
- To stop data collection, send a break and the verification characters.
- To conserve power between measurement intervals, send the command PD.

A typical Aquadopp session might look like this:

Aquadopp sends	Controller sends	Comments
	<BREAK>	Aquadopp in power down
Aquadopp Nortek AS 2003 Version 1.23 Command mode AckAck		In command mode
	AD	
AckAck		Measuring
....) (*&) (*&) (&		Outputs binary data
&!%#&)*J ASH(#&		
		Aquadopp is powered down

Chapter 4

Remote Control Commands

A few terms:

RTC: Real Time Clock
MSW: Most Significant Word, bits 31–16 in a 32 bits data field
LSW: Least Significant Word, bits 15–0 in a 32 bits data field
SW: The software program in the computer or controller
FW: The software program in the instrument
0x: Indicates hex code

Transmission is Intel style, i.e. low byte must be sent to (and will also be received from) the instrument before the high byte.

Consequently, the Hex command code in the instruction tables is shown in reversed order to reflect a 16bit word point of view rather than a per byte point of view, see also text for details.

The time delay between the two characters in a command must be less than 0.5 second, otherwise both characters will be discarded by the Paradopp.

Low byte before high byte. When designing computers, there are two different architectures for handling memory storage. They are often called Big Endian and Little Endian and refer to the order in which the bytes are stored in memory. The Windows series of operating systems has been designed around Little Endian architecture and is not compatible with Big Endian.

These two phrases are derived from “Big End In” and “Little End In.” They refer to the way in which memory is stored. On an Intel computer, the little end is stored first. This means a Hex word like 0x1234 is stored in memory as (0x34 0x12). The little end, or lower end, is stored first. The same is true for a four-byte value; for example, 0x12345678 would be stored as (0x78 0x56 0x34 0x12). For this reason we show the Hex values in reversed order in the below tables.

Example: The character ‘R’ corresponds to 0x52 and the character ‘C’ to 0x43. Shown in reversed order (to comply with the Little Endian principle) this will read 0x4352, which is what you will find listed in the table: Remote Control Commands in Command Mode

Read clock	
Execute command **RC**	Description Reads the current date and time of the RTC in the instrument.
Hex **4352**	Response 3 words clock data structure followed by AckAck
	Command parameter required None

	Response example 28 18 13 11 04 02 06 06
Reference Chapter on Data Structures	Note

Set clock

Execute command **SC** Hex **4353**	Description Sets the current date and time of the RTC in the instrument.
	Response **AckAck**
	Parameter 3 word clock data structure
Reference Chapter on Data Structures	Note The setting of the clock is synchronized to the transition of seconds i.e. the FW waits until a second transition has occurred and then sets the clock.

Inquiry

Execute command **II** Hex **4949**	Description Returns a word **z** telling which mode the instrument is in.
	Response structure 1 word **z** followed by **AckAck**
	Response interpretation z = 0x0000: Firmware upgrade mode z = 0x0001: Measurement mode z = 0x0002: Command mode z = 0x0004: Data retrieval mode z = 0x0005: Confirmation mode
	Response example 02 00 06 06 Indicating **Command mode** followed by **AckAck**
Reference	Note In measurement mode all commands are single character. This means that if you send a normal inquiry command, the instrument will return the result twice. The Inquiry command is the preferred command to use for automatic baud rate detection for the PC.

Set baud rate

Execute command **BR** Hex **5242**	Description Sets the instrument baud rate.
	Response **AckAck**
	Parameter to be sent Z
	Parameter structure z = 0x3030 300 baud z = 0x3131 600 baud z = 0x3231 1200 baud z = 0x3333 2400 baud z = 0x3434 4800 baud z = 0x3535 9600 baud z = 0x3636 19200 baud z = 0x3737 38400 baud z = 0x3838 57600 baud z = 0x3939 115200 baud z = 0x3031 600000 baud z = 0x3231 1200000 baud
	Example 5242 3232 06 06 Command for setting baud rate to 1200 baud followed by the response AckAck
Reference	Note The baud rate is stored in the instrument only when the recorder is formatted, when a measurement is started, or an SB command is sent. This will ensure that the communication can be restored by waiting until the instrument powers down after 5 minutes of inactivity. The PC must make sure that the baud rate being used is sufficiently high to ensure that all data can be transferred over the serial line for the chosen data format and measurement interval. For example, at 300 baud it is only possible to transfer 30 bytes/s, so having a measurement interval of 1 second will not be possible. Using very low baud rates will inevitably have an impact on the power consumption because of the added time needed for data transfer on the serial line before the instrument can power down. For most applications, however, the difference will be negligible. Note that baud rates above 115200 will require hardware support in the instrument and in the PC in order to work correctly.

Save baud rate	
Execute command **SB** Hex **4253**	Description Saves the currently set baud rate.
	Response **AckAck**
	Parameter **0x4733 0x3241**
Reference	Note ASCII counterpart: 3GA2. If you have set the baud with the BR command you must save it afterwards if you want the instrument to wake up with that baud rate after power down. The parameter is shown obeying the low-byte-before-high-byte principle, i.e. in the way it should be sent to the instrument.

Read complete configuration data	
Execute command **GA** Hex **4147**	Description Read the currently used hardware configuration, the head configuration, and the deployment configuration from the instrument
	Response Complete setup information (48 +224+512 bytes) followed by **AckAck**
	Parameter None
	Response example a5 05 18 00 41 51 44 20 31 32 31 35 20 20 20 20 20 20 02 00 d0 07 0d 00 3c 00 90 00 01 00 ff ff ff ff ff ff ff ff ff ff ff ff 31 2e 31 31 98 5c a5 04 70 00 0d 00 d0 07 00 00 41 51 50 20 30 38 35 32 20 20 20 00 19 00 19 00 19 00 00 00 3d 19 . . . cd ff 8b 00 e5 00 ee 00 0b 00 84 ff 3d ff 4c 52 06 06
Reference Chapter on Data Structures	Note Some lines in the above example have been removed for clarity.

Read deployment configuration data	
Execute command **GC** Hex **4347**	Description Read the currently used deployment configuration from the instrument.
	Response Deployment setup (512 bytes) followed by **AckAck**
	Parameter None
	Response example a5 00 00 01 5c 00 22 00 18 00 b6 02 00 02 0f 00 01 00 03 00 02 00 60 00 00 00 00 00 00 00 01 00 01 00 14 00 14 17 01 00 00 00 00 00 00 00 00 00 50 01 06 10 04 02 06 00 00 00 20 00 11 41 01 00 01 00 0f 00 00 00 00 00 a8 2f 5e 01 ca 3c e6 3c . . . cd ff 8b 00 e5 00 ee 00 0b 00 84 ff 3d ff 4c 52 06 06
Reference Chapter on Data Structures	Note Some lines in the above example have been removed for clarity.

Read hardware configuration data	
Execute command **GP** Hex **5047**	Description Read the currently used hardware configuration from the instrument.
	Response HW setup information (48 bytes) followed by **AckAck**
	Parameter None
	Response example a5 05 18 00 41 51 44 20 31 32 31 35 20 20 20 20 20 20 02 00 d0 07 0d 00 3c 00 90 00 01 00 ff ff ff ff ff ff ff ff ff ff ff ff 31 2e 31 31 98 5c 06 06
Reference Chapter on Data Structures	Note

Read head configuration data

Execute command	Description
GH Hex **4847**	Read the currently used head configuration from the instrument
	Response Head configuration (224 bytes) followed by **AckAck**
	Parameter None
	Response example 15 15 a5 04 70 00 0d 00 d0 07 00 00 41 51 50 20 30 38 35 32 20 20 20 00 19 00 19 00 19 00 00 00 . . . 38 0e 10 0e 10 0e 10 27 64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 00 f0 60 06 06
Reference Chapter on Data Structures	Note Some lines in the above example have been removed for clarity.

Format recorder

Execute command	Description
FO Hex **4f46**	The format recorder command formats the recorder's memory. This affects the data acquired only. Configuration files – including the deployment setup – are not affected.
	Response **AckAck**
	Parameter to be sent **0xd412 0xef1e**
Reference	Note This parameter has no ASCII counterpart. The parameter is shown obeying the low-byte-before-high-byte principle, i.e. in the way it should be sent to the instrument.

Configure instrument

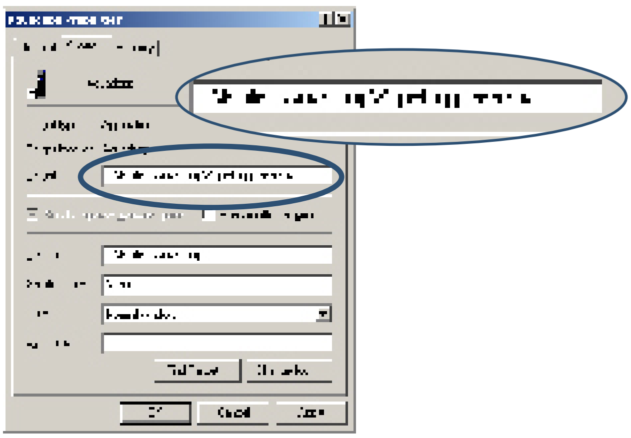
Execute command	Description
CC Hex **4343**	Use this command to download a new deployment file to the instrument.
	Response **AckAck**
	Parameter sent out The setup file to be downloaded
Reference User guide for your Nortek instrument	Note The command must be followed by a deployment setup file – how to generate this, see below.

Deployment File in Binary Format

Always use the Nortek software accompanying your Nortek instrument when making deployment files. This will save you from a lot of unneeded efforts! When you have generated the file, you may save it. However, this will not generate a file in binary format suitable for direct download to your controller.

To generate deployment files in binary format:

1. Generate a new shortcut to the Nortek software.
2. Append the characters -cu in the target line as shown below (using Aquadopp as example).



3. Start the Nortek software using the new shortcut.
4. When you now save the deployment file, this will generate two files – the regular file and a file in binary format with the file extension .pcf. This file is the one to download with your controller.

Power down

Execute command **PD** Hex **4450**	Description The Power down command puts the instrument in sleep mode (switches off the power) Response **AckAck** Parameter None

Read File Allocation Table (FAT)	
Execute command **RF** Hex **4652**	Description Reads the FAT in the instrument. Response FAT followed by **AckAck** Parameter sent out None
Reference Chapter on Data Structures	Note The FAT is a 16 byte × 32 lines matrix containing information about names and start & stop addresses of the measurement files. Up to 31 different measurement files can be stored in the instrument's recorder (the last line in the matrix is spare and not used).

Read file configuration	
Execute command **FC** Hex **4346**	Description Returns the hardware, head and deployment setup for a particular measurement file. Response Complete setup information (48+224+512 bytes) for the selected file followed by **AckAck** Parameter sent out **0x001e** using the file's location in the FAT as index
Reference GA command RF command	Note This command corresponds to the GA command, but for a specific file.

Read recorder data file

Execute command **RD** Hex **4452**	Description Returns the contents of part of the recorder memory as defined by a start and a stop address.
	Response As many bytes of data as requested followed by **AckAck**
	Parameter sent out **<Start address><Stop address>**
	Parameter example Assume you want to read in the contents from **0x00002a00** to **0x00003a00.** The following will be sent (hex values): 4452002a0000003a0000
Reference RF command	Note You will normally pick the Start address and the Stop address from the FAT when programming your controller. However, you may also use this command to transfer part of the measurement file.

Read recorder data block with CRC

Execute command **DC** Hex **4652**	Description As the RD command, but with CRC appended to the data transferred.
	Response As many bytes of data as requested followed by the CRC and **AckAck**
	Parameter sent out As for RD
Reference RD command	Note As for RD

Battery voltage

Execute command **BV** Hex **5642**	Description Read the battery voltage from the instrument.
	Response 4 bytes followed by **AckAck**
	Parameter sent out None
	Response example a7 1f 06 06
Reference	Note Bearing in mind the low-byte-before-high-byte principle, the response should be interpreted as **0x1fa7** which corresponds to decimal **8103**, which in turn is the voltage directly in mV.

Transparent compass

Execute command **TC** Hex **4354**	Description The transparent compass command powers up the compass and makes a transparent channel from the compass to the PC. Response **AckAck** Parameter None
Reference	Note This command enables the PC to read the data strings output from the compass and to send commands to the compass. Observe that this command sets the baud rate of the instrument to 38400. However, the baud rate is set back to the current instrument baud rate once a break is sent to the instrument from the controller. You can use this command to tell the controller to verify that the compass is outputting the correct data. It can also be used to set up the compass to match the required setup for the instrument. Caution! The FW command will never attempt to change the setup of the compass, so if the controller sends commands to the compass, these must set up the compass correctly before you send a break to end the transparent compass session!

Get identification string	
Execute command **ID** Hex **4449**	Description Read the identification string from the instrument. Response 14 bytes ASCII string followed by **AckAck** Parameter None Response example 41 51 44 20 31 32 31 35 20 20 20 20 20 20 06 06 corresponding to AQD1215

Start measurement without recorder	
Execute command **ST** Hex **5453**	Description Immediately starts a measurement based on the current configuration of the instrument without storing data to the recorder. Data is output on the serial port. Response **AckAck** or **NackNack** Parameter None

Reference	Note If the measurement was successfully started, **AckAck** is returned. If the measurement could not be started **NackNack** is returned. The reason for failing to start is usually that the instrument configuration is invalid.

Start measurement with recorder, at a specific time	
Execute command **SD** Hex **4453**	Description Starts a measurement at a specified time based on the current configuration of the instrument. Data is stored to a new file in the recorder. Data is output on the serial port only if specified in the configuration.
	Response **AckAck** or **NackNack**
	Parameter None
Reference	Note If the measurement was successfully started, **AckAck** is returned. If the measurement could not be started **NackNack** is returned. The reason for failing to start is usually that the instrument configuration is invalid or that the recorder is full.

Acquire data	
Execute command **AD** Hex **4441**	Description Starts a single measurement based on the current configuration of the instrument without storing data to the recorder. Instrument enters Power Down Mode when measurement has been made.
	Response **AckAck** or **NackNack**
	Parameter None
Reference	Note If the measurement was successfully started, **AckAck** is returned. If the measurement could not be started **NackNack** is returned. The reason for failing to start is usually that the instrument configuration is invalid. If the instrument is configured for continuous measurement it will keep taking data until a break is received. Upn receipt of a break it will jump directly into Command Mode.

Start measurement with recorder	
Execute command **SR** Hex **5253**	Description Immediately starts a measurement based on the current configuration of the instrument. Data is stored to a new file in the recorder and also output on the serial port
	Response **AckAck** or **NackNack**
	Parameter None
Reference	Note If the measurement was successfully started, **AckAck** is returned. If the measurement could not be started **NackNack** is returned. The reason for failing to start is usually that the instrument configuration is invalid or that the recorder is full. If the filename is not set in the configuration the filename will default to DEF.

Remote Control Commands

Data Collection Mode

Time remaining of average interval	
Execute command **A** Hex **41**	Description This command returns a word that indicates the number of seconds left of the average or burst interval.
	Response A 16-bit word followed by **AckAck**
	Parameter None
	Response example 0x1e00 0606
Reference	Note The commands available in data collection mode are all single character commands. Before sending these commands, the controller must transmit a character with binary value 0x00.

Time remaining of measurement interval	
Execute command **M** Hex **4D**	Description This command returns a word that indicates the number of seconds left of the measurement interval.
	Response A 16-bit word followed by **AckAck**
	Parameter None
	Response example 0x1c01 0606
Reference	Note The commands available in data collection mode are all single character commands. Before sending these commands, the controller must transmit a character with binary value 0x00.

Inquiry	
Execute command **I** Hex **49**	Description Returns a word z telling which mode the instrument is in.
	Response See II command
	Parameter See II command
Reference See II command	Note The commands available in data collection mode are all single character commands. Before sending these commands, the controller must transmit a character with binary value 0x00.

The commands available in data collection mode are all single character commands. Before sending these commands, the controller must transmit a character with binary value 0x00 or the character @ and then wait for 100 ms before sending the command.
The idea behind the commands in data collection mode is to allow the controller to find out where the instrument is in its measurement cycle. It is thus possible to interrogate the system without disturbing the data collection.
The inquiry (see II command) is present in all modes. In data collection mode only one character 'I' is used (see overleaf). If the standard inquiry command is sent ('II'), the system will send the response twice.

Confirmation Commands

Enter Command mode	
Execute command **MC** Hex **434d**	Description Preceded by a break command, this command is sent to force the instrument to exit Measurement mode and enter Command mode.
	Response **AckAck**
	Parameter None
Reference	Note The MC command must be sent within 10 seconds after the break was sent. Otherwise the measurement will continue. Within 2 seconds after **AckAck** is sent, the instrument will enter Command mode

Chapter 5

Firmware Data Structures

This section describes the data structures that are used for the Paradopp products. They are grouped in generic data structures that are common to all instruments and instrument specific structures.

The following firmware data structures are described:

- Generic Structures
- Aquadopp-specifc Structures
- Vector-specific Structures
- Aquadopp Profiler-specific Structures
- AWAC-specific Structures
- Continental

BCD format. *The BCD format is a a way of representing decimal figures in a binary manner. Four bits are used per digit.*
Example *(using clock data):*
10 00 05 11 04 02
(all values shown in hex) corresponds to:
5 February 2004 11:10:00.

Generic structures

Clock Data

Size	Name	Offset	Description
1	Minute	0	minute (BCD format)
1	Second	1	second (BCD format)
1	Day	2	day (BCD format)
1	Hour	3	hour (BCD format)
1	Year	4	year (BCD format)
1	Month	5	month (BCD format)
Total Size 6 Bytes			

Hardware Configuration

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	05 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
14	SerialNo	4	instrument type and serial number

2	Config	18	board configuration: bit 0: Recorder installed (0=no, 1=yes) bit 1: Compass installed (0=no, 1=yes)
2	Frequency	20	board frequency [kHz]
2	PICversion	22	PIC code version number
2	HWrevision	24	Hardware revision
2	RecSize	26	Recorder size (*65536 bytes)
2	Status	28	status: bit 0: Velocity range (0=normal, 1=high)
12	Spare	30	spare
4	FWversion	42	firmware version
2	Checksum	46	= b58c(hex) + sum of all bytes in structure
Total Size 48 Bytes			

Head Configuration

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	04 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Config	4	head configuration: bit 0: Pressure sensor (0=no, 1=yes) bit 1: Magnetometer sensor (0=no, 1=yes) bit 2: Tilt sensor (0=no, 1=yes) bit 3: Tilt sensor mounting (0=up, 1=down)
2	Frequency	6	head frequency (kHz)
2	Type	8	head type
12	SerialNo	10	head serial number
176	System	22	system data
22	Spare	198	spare
2	NBeams	220	number of beams
2	Checksum	222	= b58c(hex) + sum of all bytes in structure
Total Size 224 Bytes			

User Configuration

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	00 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	T1	4	transmit pulse length (counts)
2	T2	6	blanking distance (counts)
2	T3	8	receive length (counts)
2	T4	10	time between pings (counts)
2	T5	12	time between burst sequences (counts)
2	NPings	14	number of beam sequences per burst
2	AvgInterval	16	average interval in seconds
2	NBeams	18	number of beams

2	TimCtrlReg	20	timing controller mode **bit 1:** profile (0=single, 1=continuous) **bit 2:** mode (0=burst, 1=continuous) **bit 5:** power level (0=1, 1=2, 0=3, 1=4) **bit 6:** power level (0 0 1 1) **bit 7:** synchout position (0=middle of sample, 1=end of sample (Vector)) **bit 8:** sample on synch (0=disabled,1=enabled, rising edge) **bit 9:** start on synch (0=disabled,1=enabled, rising edge)
2	PwrCtrlReg	22	power control register **bit 5:** power level (0=1, 1 =2, 0=3, 1=4) **bit 6:** power level (0 0 1 1)
2	A1	24	not used
2	B0	26	not used
2	B1	28	not used
2	CompassUpdRate	30	compass update rate
2	CoordSystem	32	coordinate system (0=ENU, 1=XYZ, 2=BEAM)
2	NBins	34	number of cells
2	BinLength	36	cell size
2	MeasInterval	38	measurement interval
6	DeployName	40	recorder deployment name
2	WrapMode	46	recorder wrap mode (0=NO WRAP, 1=WRAP WHEN FULL)
6	clockDeploy	48	deployment start time
4	DiagInterval	54	number of seconds between diagnostics measurements
2	Mode	58	mode: **bit 0:** use user specified sound speed (0=no, 1=yes) **bit 1:** diagnostics/wave mode 0=disable, 1=enable) **bit 2:** analog output mode (0=disable, 1=enable) **bit 3:** output format (0=Vector, 1=ADV) **bit 4:** scaling (0=1 mm, 1=0.1 mm) **bit 5:** serial output (0=disable, 1=enable) **bit 6:** reserved EasyQ **bit 7:** stage (0=disable, 1=enable) **bit 8:** output power for analog input (0=disable, 1=enable)
2	AdjSoundSpeed	60	user input sound speed adjustment factor
2	NSampDiag	62	# samples (AI if EasyQ) in diagnostics mode
2	NBeamsCellDiag	64	# beams / cell number to measure in diagnostics mode
2	NPingsDiag	66	# pings in diagnostics/wave mode
2	ModeTest	68	mode test: **bit 0:** correct using DSP filter (0=no filter, 1=filter) **bit 1:** filter data output (0=total corrected velocity,1=only correction part)
2	AnaInAddr	70	analog input address
2	SWVersion	72	software version
2	Spare	74	spare
180	VelAdjTable	76	velocity adjustment table
180	Comments	256	file comments
2	Mode	436	wave measurement mode **bit 0:** data rate (0=1 Hz, 1=2 Hz) **bit 1:** wave cell position (0=fixed, 1=dynamic) **bit 2:** type of dynamic position (0=pct of mean pressure, 1=pct of min re)
2	DynPercPos	438	percentage for wave cell positioning (=32767×#%/100) (# means number of)
2	T1	440	wave transmit pulse
2	T2	442	fixed wave blanking distance (counts)
2	T3	444	wave measurement cell size
2	NSamp	446	number of diagnostics/wave samples

2	A1	448	not used
2	B0	450	not used
2	B1	452	not used
2	Spare	454	spare
2	AnaOutScale	456	analog output scale factor (16384=1.0, max=4.0)
2	CorrThresh	458	correlation threshold for resolving ambiguities
2	Spare	460	spare
2	TiLag2	462	transmit pulse length (counts) second lag
30	Spare	464	spare
16	QualConst	494	stage match filter constants (EZQ)
2	Checksum	510	=b58c(hex)+sum of all bytes in structure
Total Size 512 Bytes			

File Allocation Table

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15
File name (ASCII) of file #0						Seq	Status	Start address				Stop address			
File name (ASCII) of file #1						Seq	Status	Start address				Stop address			
File name (ASCII) of file #2						Seq	Status	Start address				Stop address			
⋮															
File name (ASCII) of file #29						Seq	Status	Start address				Stop address			
File name (ASCII) of file #30						Seq	Status	Start address				Stop address			
Not used															

Seq: If several files share the same file name, they must be distinguished by their position in the FAT. The standard instrument software has implemented this by appending a sequence number to the file name. Example: Multiple use of the file name ANTHON will produce ANTHON1, ANTHON2 etc., in which 1 and 2 are the contents of the Sequence byte (added automatically).

Status: If bit 0 (the LSB) has been set to 1, file wrapping has been enabled. If bit 1 has been set to 1, a complete wrap-around has occurred, i.e. all the data initially stored has been overwritten at least once.

Start address: The start address of the measured data

Stop address: The stop address of the measured data

Altogether a maximum of 31 different measurement files may be stored in the instrument's internal recorder. The length of these files depends on the amount of memory installed.

Aquadopp specific structures

Aquadopp Velocity Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	01 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	Error	10	error code
2	AnaIn1	12	analog input 1
2	Battery	14	battery voltage (0.1 V)
2	SoundSpeed/AnaIn2	16	speed of sound (0.1 m/s) or analog input 2
2	Heading	18	compass heading (0.1°)
2	Pitch	20	compass pitch (0.1°)
2	Roll	22	compass roll (0.1°)
1	PressureMSB	24	pressure MSB (mm) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	status code
2	PressureLSW	26	pressure LSW (mm) (Pressure = 65536×PressureMSB + PressureLSW)
2	Temperature	28	temperature (0.01 °C)
2	Vel B1/X/E	30	velocity beam1 or X or East coordinates (mm/s)
2	Vel B2/Y/N	32	velocity beam2 or Y or North coordinates (mm/s)
2	Vel B3/Z/U	34	velocity beam3 or Z or Up coordinates (mm/s)
1	Amp B1	36	amplitude beam1 (counts)
1	Amp B2	37	amplitude beam2 (counts)
1	Amp B3	38	amplitude beam3 (counts)
1	Fill	39	fill byte
2	Checksum	40	= b58c(hex) + sum of all bytes in structure
Total Size 42 Bytes			

Aquadopp Diagnostics Data Header

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	06 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Records	4	number of diagnostics samples to follow
2	Cell	6	cell number of stored diagnostics data
1	Noise1	8	noise amplitude beam 1 (counts)
1	Noise2	9	noise amplitude beam 2 (counts)
1	Noise3	10	noise amplitude beam 3 (counts)
1	Noise4	11	noise amplitude beam 4 (counts)

2	ProcMagn1	12	processing magnitude beam 1
2	ProcMagn2	14	processing magnitude beam 2
2	ProcMagn3	16	processing magnitude beam 3
2	ProcMagn4	18	processing magnitude beam 4
2	Distance1	20	distance beam 1
2	Distance2	22	distance beam 2
2	Distance3	24	distance beam 3
2	Distance	26	distance beam 4
6	Spare	28	spare
2	Checksum	34	= b58c(hex) + sum of all bytes in structure
Total Size 36 Bytes			

Aquadopp Diagnostics Data

Same as Aquadopp Velocity Data, except Id = 0x80.

Vector specific structures

Vector Velocity Data Header

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	12 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	NRecords	10	number of velocity samples to follow
1	Noise1	12	noise amplitude beam 1 (counts)
1	Noise2	13	noise amplitude beam 2 (counts)
1	Noise3	14	noise amplitude beam 3 (counts)
1	Noise4	15	noise amplitude beam 4 (counts)
1	Correlation	16	noise correlation beam 1
1	Correlation	17	noise correlation beam 2
1	Correlation	18	noise correlation beam 3
1	Correlation	19	noise correlation beam 4
20	Spare	20	spare
2	Checksum	40	= b58c(hex) + sum of all bytes in structure
Total Size 42 Bytes			

Vector Velocity Data

Size	Name	Offset	Description

1	Sync	0	a5 (hex)
1	Id	1	10 (hex)
1	AnaIn2LSB	2	analog input 2 LSB
1	Count	3	ensemble counter
1	PressureMSB	4	pressure MSB (mm) (Pressure = 65536×PressureMSB + PressureLSW)
1	AnaIn2MSB	5	analog input 2 MSB
2	PressureLSW	6	pressure LSW (mm) (Pressure = 65536×PressureMSB + PressureLSW)
2	AnaIn1	8	analog input 1
2	Vel B1/X/E	10	velocity beam1 or X or East (mm/s)
2	Vel B2/Y/N	12	velocity beam2 or Y or North (mm/s)
2	Vel B3/Z/U	14	velocity beam3 or Z or Up (mm/s)
1	Amp B1	16	amplitude beam1 (counts)
1	Amp B2	17	amplitude beam2 (counts)
1	Amp B3	18	amplitude beam3 (counts)
1	Corr B1	19	correlation beam1 (%)
1	Corr B2	20	correlation beam2 (%)
1	Corr B3	21	correlation beam3 (%)
2	Checksum	22	= b58c(hex) + sum of all bytes in structure
Total Size 24 Bytes			

Vector System Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	11 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	Battery	10	battery voltage (0.1 V)
2	SoundSpeed	12	speed of sound (0.1 m/s)
2	Heading	14	compass heading (0.1 deg)
2	Pitch	16	compass pitch (0.1 deg)
2	Roll	18	compass roll (0.1 deg)
2	Temperature	20	temperature (0.01 deg C)
1	Error	22	error code
1	Status	23	status code
2	AnaIn	24	analog input
2	Checksum	26	= b58c(hex) + sum of all bytes in structure
Total Size 28 Bytes			

Aquadopp Profiler specific structures

Aquadopp Profiler Velocity Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	21 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	Error	10	error code
2	AnaIn1	12	analog input 1
2	Battery	14	battery voltage (0.1 V)
2	SoundSpeed/AnaIn2	16	speed of sound (0.1 m/s) or analog input 2
2	Heading	18	compass heading (0.1°)
2	Pitch	20	compass pitch (0.1°)
2	Roll	22	compass roll (0.1°)
1	PressureMSB	24	pressure MSB (mm) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	status code
2	PressureLSW	26	pressure LSW (mm) (Pressure = 65536×PressureMSB + PressureLSW)
2	Temperature	28	temperature (0.01 °C)
2	Vel 1 B1/X/E	30	velocity cell 1, beam1 or X or East (mm/s)
2..	Vel 2···n	32···	...repeated for cells 2 through n
2	Vel 1 B2/Y/N		velocity cell 1, beam2 or Y or North (mm/s)
2..	Vel 2···n		...repeated for cells 2 through n
2	Vel 1 B3/Z/U		velocity cell 1, beam3 or Z or Up (mm/s)
2..	Vel 2···n		...repeated for cells 2 through n
1	Amp 1 B1		amplitude cell 1, beam1 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Amp 1 B2		amplitude cell 1, beam2 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Amp 1 B3		amplitude cell 1, beam3 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Fill		fill byte if number of cells mod 2 is not equal to 0
2	Checksum		= b58c(hex) + sum of all bytes in structure
Total Size is variable			

AWAC specific structures

Awac Velocity Profile Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	20 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	Error	10	error code
2	AnaIn1	12	analog input 1
2	Battery	14	battery voltage (0.1 V)
2	SoundSpeed/AnaIn2	16	speed of sound (0.1 m/s) or analog input 2
2	Heading	18	compass heading (0.1°)
2	Pitch	20	compass pitch (0.1°)
2	Roll	22	compass roll (0.1°)
1	PressureMSB	24	pressure MSB (mm) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	status code
2	PressureLSW	26	pressure LSW (mm) (Pressure = 65536×PressureMSB + PressureLSW)
2	Temperature	28	temperature (0.01 °C)
88	Spare	30	spare
2	Vel 1 B1/X/E	118	velocity cell 1, beam1 or X or East (mm/s)
2..	Vel 2···n	120	...repeated for cells 2 through n
2	Vel 1 B2/Y/N		velocity cell 1, beam2 or Y or North (mm/s)
2..	Vel 2···n		...repeated for cells 2 through n
2	Vel 1 B3/Z/U		velocity cell 1, beam3 or Z or Up (mm/s)
2..	Vel 2···n		...repeated for cells 2 through n
1	Amp 1 B1		amplitude cell 1, beam1 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Amp 1 B2		amplitude cell 1, beam2 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Amp 1 B3		amplitude cell 1, beam3 (counts)
1..	Amp 2···n		...repeated for cells 2 through n
1	Fill		fill byte if number of cells mod 2 is not equal to 0
2	Checksum		= b58c(hex) + sum of all bytes in structure
Total Size is variable			

Awac Wave Data Header

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	31 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
1	Minute	4	minute (BCD)
1	Second	5	second (BCD)
1	Day	6	day (BCD)
1	Hour	7	hour (BCD)
1	Year	8	year (BCD)
1	Month	9	month (BCD)
2	NRecords	10	number of wave data records to follow
2	Blanking	12	blanking distance (counts)
2	Battery	14	battery voltage (0.1 V)
2	SoundSpeed	16	speed of sound (0.1 m/s)
2	Heading	18	compass heading (0.1°)
2	Pitch	20	compass pitch (0.1°)
2	Roll	22	compass roll (0.1°)
2	MinPress	24	min pressure value of previous profile (mm)
2	hMaxPress	26	max pressure value of previous profile (mm)
2	Temperature	28	temperature (0.01 °C)
2	CellSize	30	cell size in counts of T3
1	Noise1	32	noise amplitude beam 1 (counts)
1	Noise2	33	noise amplitude beam 2 (counts)
1	Noise3	34	noise amplitude beam 3 (counts)
1	Noise4	35	noise amplitude beam 4 (counts)
2	ProcMagn1	36	processing magnitude beam 1
2	ProcMagn2	38	processing magnitude beam 2
2	ProcMagn3	40	processing magnitude beam 3
2	ProcMagn4	42	processing magnitude beam 4
14	Spare	44	spare
2	Checksum	58	= b58c(hex) + sum of all bytes in structure
Total Size is 60 Bytes			

Awac Wave Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	30 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Pressure	4	pressure (mm)
2	Distance	6	distance to vertical beam
2	AnaIn	8	analog input
2	Vel1	10	velocity beam 1 (mm/s)
2	Vel2	12	velocity beam 2 (mm/s)
2	Vel3	14	velocity beam 3 (mm/s)
2	Vel4	16	velocity beam 4 (mm/s)
1	Amp1	18	amplitude beam 1 (mm/s)

1	Amp2	19	amplitude beam 2 (mm/s)
1	Amp3	20	amplitude beam 3 (mm/s)
1	Amp4	21	amplitude beam 4 (mm/s)
2	Checksum	22	= b58c(hex) + sum of all bytes in structure
Total Size is 24 Bytes			

Continental Data

Same as AWAC Profiler Data, except ID = 0x24

Prolog

The Prolog is a module that can be added to Nortek instruments in replacement of the standard recorder, where it is inserted in the location of the recorder. The pure recorder variant uses an industrial grade SD card with 4 GB of memory. The industrial grade SD card is sealed and watertight, which means that it keeps with the Nortek philosophy that the data should still be available in the event of damage to the instrument that leads to a leak.

The Prolog has additional functionality of wave processing when used with an AWAC. Data may be streamed out the serial line of the AWAC in either binary format or as NMEA ASCII strings. When wave processing is enabled the processed data – as well as raw data - is stored on the SD card. Below is a detailed description of both the binary data formats and the NMEA data strings that the user would need to either convert or parse.

Note that the AWAC software includes the conversion of the processed binary data file (*.WPB) found on the SD card. This is found under the ASCII data conversion tool

More information about the use and description of the data products is found in instrument deployment software (e.g., AWAC AST software).

Prolog specific structures

The recorded processed, binary data is composed of the instruments header data structures (User, Head, Hardware, etc.), current profile data structure, and all of the processed wave data enabled by the user. The wave parameter data structure (PdWaveData) is always included in processed wave data structures. The following is a description of the processed wave data structures

Wave parameter estimates

Size	Name	Offset	Description
1	cSync		A5 (hex)
1	cID		60 (hex)
2	Size		size in words
6	clock		date and time
1	hSpectrumTyp		spectrum used for calculation
1	hProcMethod		processing method used in actual calculation

2	Hm0		Spectral significant wave height [mm]
2	H3		AST significant wave height (mean of largest 1/3) [mm]
2	H10		AST wave height(mean of largest 1/10) [mm]
2	Hmax		AST max wave height in wave ensemble [mm]
2	Tm02		Mean period spectrum based [0.01 sec]
2	Tp		Peak period [0.01 sec]
2	Tz		AST mean zero-crossing period [0.01 sec]
2	DirTp		Direction at Tp [0.01 deg]
2	SprTp		Spreading at Tp [0.01 deg]
2	DirMean		Mean wave direction [0.01 deg]
2	UI		Unidirectivity index [1/65535]
4	hPressureMean		Mean pressure during burst [dbar/1000]
2	NumNoDet		Number of ST No detects [
2	NumBadDet		Number of ST Bad detects [
2	CurSpeedMean		Mean current speed - wave cells [mm/sec]
2	CurDirMean		Mean current direction - wave cells [0.01 deg]
4	hError		Error Code for bad data
28	Spares		
2	Checksum		checksum
Total Size is 80 Bytes			

Wave band estimates

Size	Name	Offset	Description
1	cSync		A5 (hex)
1	cID		61 (hex)
2	Size		size in words
6	clock		date and time
1	hSpectrumType		spectrum used for calculation
1	hProcMethod		processing method used in actual calculation
2	LowFrequency		low frequency in [0.001 Hz]
2	HighFrequency		high frequency in [0.001 Hz]
2	Hm0		Spectral significant wave height [mm]
2	Tm02		Mean period spectrum based [0.01 sec]
2	Tp		Peak period [0.01 sec]
2	DirTp		Direction at Tp [0.01 deg]
2	DirMean		Mean wave direction [0.01 deg]
2	SprTp		Spreading at Tp [0.01 deg]
4	hError		Error Code for bad data
14	Spares		
2	Checksum		checksum
Total Size is 48 Bytes			

Wave energy spectrum

Size	Name	Offset	Description
1	cSync		A5 (hex)
1	cID		62 (hex)
2	Size		size in words
6	clock		date and time
1	cSpectrumType		spectrum used for calculation
1	cSpare		
2	hNumSpectrum		Number of spectral bins (default 98)
2	LowFrequency		low frequency in [0.001 Hz]
2	HighFrequency		high frequency in [0.001 Hz]
2	StepFrequency		frequency step in [0.001 Hz]
18	Spares		
4	hEnergyMultiplier		AST energy spectrum multiplier [cm^2/Hz]
196	Energy		AST Spectra [0 - 1/65535] -
2	Checksum		checksum
Total Size is 240 Bytes			

Wave fourier coefficient spectrum

Size	Name	Offset	Description
1	cSync		A5 (hex)
1	cID		63 (hex)
2	Size		size in words
6	clock		date and time
1	cSpare		
1	cProcMethod		processing method used in actual calculation
2	NumSpectrum		Number of spectral bins (default 49)
2	LowFrequency		low frequency in [0.001 Hz]
2	HighFrequency		high frequency in [0.001 Hz]
2	StepFrequency		frequency step in [0.001 Hz]
10	Spares		
196	A1		Fourier coefficients in [+/- 1/32767]
196	B1		
196	A2		
196	B2		
2	Checksum		checksum
Total Size is 816 Bytes			

Data which is determined to be invalid in the wave processing is flagged with a hex value of 0xffff. This means that the value for unsigned is 65636 and it is -32768 for signed values

Vectrino specific structures

Vectrino velocity data header

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	50 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Distance	4	distance (0.1 mm)
2	DistQuality	6	distance quality (-1536 to +1536)
2	Lag1	8	lag1 used
2	Lag2	10	lag2 used
1	Noise1	12	noise amplitude beam 1 (counts)
1	Noise2	13	noise amplitude beam 2 (counts)
1	Noise3	14	noise amplitude beam 3 (counts)
1	Noise4	15	noise amplitude beam 4 (counts)
1	Correlation	16	noise correlation beam 1 (%)
1	Correlation	17	noise correlation beam 2 (%)
1	Correlation	18	noise correlation beam 3 (%)
1	Correlation	19	noise correlation beam 4 (%)
2	Temperature	20	temperature (0.01 deg C)
2	SoundSpeed	22	speed of sound (0.1 m/s)
1	AmpZ0	24	amplitude in sampling volume beam 1 (counts)
1	AmpZ0	25	amplitude in sampling volume beam 2 (counts)
1	AmpZ0	26	amplitude in sampling volume beam 3 (counts)
1	AmpZ0	27	amplitude in sampling volume beam 4 (counts)
1	AmpX1	28	amplitude at boundary beam 1 (counts)
1	AmpX1	29	amplitude at boundary beam 2 (counts)
1	AmpX1	30	amplitude at boundary beam 3 (counts)
1	AmpX1	31	amplitude at boundary beam 4 (counts)
1	AmpZ0PLag1	32	Z0 plus lag1 used beam 1 (counts)
1	AmpZ0PLag1	33	Z0 plus lag1 used beam 2 (counts)
1	AmpZ0PLag1	34	Z0 plus lag1 used beam 3 (counts)
1	AmpZ0PLag1	35	Z0 plus lag1 used beam 4 (counts)
1	AmpZ0PLag2	36	Z0 plus lag2 used beam 1 (counts)
1	AmpZ0PLag2	37	Z0 plus lag2 used beam 2 (counts)
1	AmpZ0PLag2	38	Z0 plus lag2 used beam 3 (counts)
1	AmpZ0PLag2	39	Z0 plus lag2 used beam 4 (counts)
2	Checksum	40	= b58c(hex) + sum of all bytes in structure
Total Size is 42 Bytes			

Vectrino velocity data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	51 (hex)
1	Status	2	[exvcccbb] status bits, where
			bb = #beams - 1
			ccc = #cells - 1
			v = velocity scaling (0 = mm/s, 1 = 0.1mm/s)
			x = not used
			e = error (0 = no error, 1 = error condition)
1	Count	3	ensemble counter (0 - 255)
2	Vel 1 B1/X	4	velocity cell 1, beam1 or X (mm/s)
2	Vel 2…n	6	…repeated for cells 2 through n
2	Vel 1 B2/Y		velocity cell 1, beam2 or Y (mm/s)
2	Vel 2…n		…repeated for cells 2 through n
2	Vel 1 B3/Z		velocity cell 1, beam3 or Z (mm/s)
2	Vel 2…n		…repeated for cells 2 through n
2	Vel 1 B4/Z2		velocity cell 1, beam4 or Z2 (mm/s)
2	Vel 2…n		…repeated for cells 2 through n
1	Amp 1 B1		amplitude cell 1, beam1 (counts)
1	Amp 2…n		…repeated for cells 2 through n
1	Amp 1 B2		amplitude cell 1, beam2 (counts)
1	Amp 2…n		…repeated for cells 2 through n
1	Amp 1 B3		amplitude cell 1, beam3 (counts)
1	Amp 2…n		…repeated for cells 2 through n
1	Amp 1 B4		amplitude cell 1, beam4 (counts)
1	Amp 2…n		…repeated for cells 2 through n
1	Corr 1 B1		correlation cell 1, beam1 (%)
1	Corr 2…n		…repeated for cells 2 through n
1	Corr 1 B2		correlation cell 1, beam2 (%)
1	Corr 2…n		…repeated for cells 2 through n
1	Corr 1 B3		correlation cell 1, beam3 (%)
1	Corr 2…n		…repeated for cells 2 through n
1	Corr 1 B4		correlation cell 1, beam4 (%)
1	Corr 2…n		…repeated for cells 2 through n
2	Checksum		= b58c(hex) + sum of all bytes in structure
Total Size is variable			

Vectrino distance data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	02 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Temperature	4	temperature (0.01 deg C)
2	SoundSpeed	6	speed of sound (0.1 m/s)
2	Distance	8	distance (0.1 mm)
2	DistQuality	10	distance quality (-1536 to +1536)
2	Spare	12	spare
2	Checksum	14	= b58c(hex) + sum of all bytes in structure
Total Size is 16 Bytes			

Chapter 6
ASCII Output

ASCII output is available for the Aquadopp single point current meter, the Aquadopp Profiler and for the Continental current profiler.

Aquadopp ASCII Output

The Aquadopp single point current meter can also output data in ASCII format. To use the ASCII output feature you must upgrade your Aquadopp firmware to version 1.13 or higher. You can download new firmware from the support pages of www.nortek-as.com.

The output format is based on the current output of the ASCII conversion with our software. All positions are the same, the only differences are that the error and status codes are output as decimal numbers (e.g. 17710 instead of 101100012), and the field de-limiter is always just a single space. The description of the format is found in the .hdr file that is generated when you convert an .aqd data file to ASCII – shown overleaf.

There are three ways to enable the ASCII output:

1. The command AS (AsciiStart) is the ASCII equivalent to the regular ST command. It starts a measurement with the current configuration and outputs the data in ASCII format. To get back into command mode you must send the confirmation characters MC after sending a break.
2. The command MA (MeasureAscii) makes one measurement with the current configuration (unless when configured for continuous measurement – see below) and outputs the data in ASCII format. There is a new binary equivalent to this command, AD (AquireData). If you want to control the data timing from a data logger you should use one of these commands. By using either the MA or the AD command, the instrument will automatically power down after the measurement is finished. Sending a break will cause the instrument to enter command mode directly, for example, if you want to stop a continuous measurement.
3. The command RA (RecorderAscii) is the ASCII equivalent to the regular SR command. It starts a measurement with the current configuration and outputs the data in ASCII format while at the same time storing the data to the internal recorder. To get back into command mode you must send the confirmation characters MC after sending a break.

The following should be observed:

- ASCII commands consist of a pair of ASCII characters, sent when the instrument is in Command mode. No other characters are required (i.e. checksum or end of line characters).

- Make sure you have configured the instrument correctly before using the ASCII output commands. To use the ASCII commands, you will first configure the instrument from the Aquadopp software by entering the required setup parameters and starting a measurement.
- Note that for the MA command to make only a single measurement, the current meter cannot be in Continuous mode. This means that it must have a measuring interval that is at least 4s longer than its averaging interval.
- When you stop the measurement to enter Command mode, the instrument will remember the last configuration, even when power is removed.
- Note to store data to the internal recorder when data is output in ASCII format you must use the RA command.

The ASCII Format

The output format is the same as the standard output of the ASCII conversion using the Aquadopp software, with the exception that speed and direction are not included. The sequence is the same, but the error and status codes are decimal numbers instead of binary (i.e. 17710 instead of 101100012) Also, the field de-limiter is always just a single space. The description of the format is found in the .hdr file that is generated when you convert an .aqd data file to ASCII.

Name	Units	Size (characters)
Month	(1-12)	2
Day	(1-31)	2
Year		4
Hour	(0-23)	2
Minute	(0-59)	2
Second	(0-59)	2
Error Code		
Status Code		
Velocity (Beam1/X/East)	m/s	5
Velocity (Beam2/Y/North)	m/s	5
Velocity (Beam3/Z/Up)	m/s	5
Amplitude (Beam1)	counts	4
Amplitude (Beam2)	counts	4
Amplitude (Beam3)	counts	4
Battery voltage	V	4
Soundspeed	m/s	6
Heading	degrees	5
Pitch	degrees	5
Roll	degrees	5
Pressure	m	7
Temperature	degrees C	4
Analogue input 1		
Analogue input 2		
Speed	m/s	
Direction	degrees	5

Here is an example with three lines of data (sampled at 10 minutes intervals):

```
4 1 2004 11 3 40 0 172 -0.104 -0.226 0.061 21 20 20 13.3 1525.5 350.5 52.9 -53.9 1.552 22.48 6304 6287
4 1 2004 11 3 44 0 172 -0.375 -0.366 -0.082 21 20   20 13.3 1525.5 350.5 52.9 -53.9 1.560 22.48 6280 6270
4 1 2004 11 3 48 0 172 -0.705 -0.526 -0.359 21 20 20 13.3 1525.5 350.3 52.9 -53.9 1.550 22.48 6204 6205
```

Aquadopp Profiler ASCII Output

The Continental is also capable of sending out ASCII formatted data.

To start a measurement with output in ASCII format the following steps must be used:
1. Set the relevant deployment parameters using the Continental software that is shipped with the instrument.
2. Download the deployment configuration to the instrument by starting a measurement.
3. Stop the measurement, the instrument has now stored the configuration internally.
4. Start an ASCII measurement from the terminal emulator using the two character command AS (Ascii Start)
5. Stop the measurement using Stop Data Collection in the Continental SW. Alternatively, the measurement can be stopped by sending a soft break followed by the characters MC (Mode Command).

Observe that there is no storage of data to the internal recorder when data are output in ASCII format.

The format is as follows:

Header Line

Size (characters)	Name
12	Serial No
1	Separator
4	Year
1	Separator
2	Month
1	Separator
2	Day
1	Year
1	Separator
2	Hour
1	Separator
2	Min
1	Separator
2	Second
1	Separator
6	Temperature
1	Separator
7	Extra field

Data Line

Size (characters)	Name
3	CellNo
1	Separator
6	Speed
1	Separator
4	Direction

Prolog ASCII Serial Ouput

The serial output may be either binary or ASCII, but not both. The current profile and sensor data is always streamed out amongst the data structures when serial output is activated. The user can select the different processed data types output when wave processing is enabled; wave parameters are always output.

When the current profile is to be followed by a wave measurement, the profile data is output together with the processed wave data. Otherwise the current profile is output at the end of the average interval.

It is possible to use the Prolog in an emulation mode where a raw AWAC data file (*.WPR) is re-processed. This can be done either using the configuration in the WPR-file on the SD card or using the configuration set with the AWAC software. These two emulations of wave processing and data streaming is started with the commands EWFF (Emulate Wave File configuration) and EWSS (Emulate Wave Software Setup), respectively. When either of these commands are sent to the AWAC (in a terminal emulator program, such as the AWAC's), the ProLog will open the first WPR-file it finds. If serial output is enabled there will be a 10 second delay before the processing is started to allow connection to another serial port. Data are, as during regular measurements, always output to file and also output on the serial port according to the configuration (NMEA, binary, SeaState online format). If the SeaState online format is used the station configuration is output before the wave processing starts.

ASCII serial data is output according to NMEA standard. The format does not follow the NMEA standard strictly (no limits on length), but uses this standard as the bases for comma separated data. The following is a description of how the different data are formatted.

Information (configuration)

Field	Size	Description	Form
0		Identifier	"$PNORI"
1		Instrument type	n
2		Head ID	aaannnn
3		Number of beams	n
4		Number of cells	n
5		Blanking (m)	dd.d
6		Cell size (m)	dd.d
7		Coordinate system	n
8		Checksum	*hh

Example:

$PNORI,3,WAV1234,3,20,0.5,5.0,0*6A

Sensor data

Field	Size	Description	Form
0		Identifier	"$PNORS"
1		Date	MMDDYY
2		Time	hhmmss.s
3		Error code (hex)	hh
4		Status code (hex)	hh
5		Battery voltage (V)	dd.d
6		Sound speed (m/s)	ddd.d
7		Heading (deg)	ddd.d
8		Pitch (deg)	dd.d
9		Roll (deg)	dd.d
10		Pressure (dbar)	ddd.ddd
11		Temperature (deg C)	dd.dd
12		Analog input #1 (counts)	nnn
13		Analog input #2 (counts)	nnn
14		Checksum (hex)	*hh

Example (empty fields = not used):

$PNORS, 082709,093913,00,0D,16.2,15.2,1490.9,182.2,-0.5,1.6,0.211,11.34,,*45

Current velocity data

Field	Size	Description	Form
0		Identifier	"$PNORC"
1		Date	MMDDYY
2		Time	hhmmss.s
3		Cell number	n
4		Velocity 1 (m/s)	dd.dd
5		Velocity 2 (m/s)	dd.dd
6		Velocity 3 (m/s)	dd.dd
7		Speed (m/s)	dd.dd
8		Direction (deg)	ddd.d
9		Amplitude units	"C" counts
10		Amplitude 1	nnn
11		Amplitude 2	nnn
12		Amplitude 3	nnn
13		Correlation 1 (%)	nn
14		Correlation 2 (%)	nn
15		Correlation 3 (%)	nn
16		Checksum (hex)	*hh

Example (empty fields = not used):

$PNORC,1,1.2,0.3,1.2,67,85,77,,,*E3
$PNORC,2,1.1,0.2,1.1,60,84,76,,,*23
$PNORC,3,1.2,0.3,1.4,64,82,76,,,*34

Wave parameters

Field	Size	Description	Form
0		Identifier	"$PNORW"
1		Date	MMDDYY
2		Time	hhmmss.s
3		Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	n
4		Processing method (0, 1, 2, 3)	n
5		Hm0 (m)	dd.dd
6		H3 (m)	dd.dd
7		H10 (m)	dd.dd
8		Hmax (m)	dd.dd
9		Tm02 (s)	dd.dd
10		Tp (s)	dd.dd
11		Tz (s)	dd.dd
12		DirTp (deg)	dd.d
13		SprTp (deg)	dd.d
14		Main Direction (deg)	dd.d
15		Unidirectivity Index	nnn
16		Mean pressure (dbar)	dd.dd
17		Number of no detects	n
18		Number of bad detects	n
19		Near surface Current speed (m/s)	dd.dd
20		Near surface Current direction (deg)	dd.d
21		Error Code	hh
22		Checksum (hex)	*hh

Example:
$PNORW,082709,093913,0,0.01,-99999,0.01,0.01,6.15,5.93, -999,326.57, 77.42,45.85,0.38,6.08,2,248,0.37,52.66,1030,*FD

Where -999 is the value used for invalid data.

Wave energy density spectrum

Field	Size	Description	Form
0		Identifier	"$PNORE"
1		Date	MMDDYY
2		Time	hhmmss.s
3		Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	n
4		Start Frequency (Hz)	d.dd

5		Step Frequency (Hz)	d.dd
6		Number of Frequencies N	nn
7		Energy Density [frequency 1] (cm2/Hz)	dddd.dd
8		Energy Density [frequency 2] (cm2/Hz)	dddd.dd
N+6		Energy Density [frequency N] (cm2/Hz)	dddd.dd
N+7		Checksum (hex)	*hh

Example:
$PNORE,082709,093913,3,0.01,0.01,49,0.0421,0.353,6.154,100.35,434.12,….,*AD

Fourier coefficient spectra

Field	Size	Description	Form
0		Identifier	"$PNORF"
1		Fourier coefficient flag [A1/B1/A2/B2]	"CC"
2		Date	MMDDYY
3		Time	hhmmss.s
4		Processing method (0, 1, 2, 3)	n
5		Start Frequency (Hz)	d.dd
6		Step Frequency (Hz)	d.dd
7		Number of Frequencies N	nn
8		Fourier Coefficient CC [frequency 1]	d.dddd
9		Fourier Coefficient CC [frequency 2]	d.dddd
N+7		Fourier Coefficient CC [frequency N]	d.dddd
N+8		Checksum (hex)	*hh

Example:
$PNORF,A1,082709,093913,3,0.01,0.01,49,0.044,0.124,0.215,0.399,0.524,….,*FE

Wave band parameters

Field	Size	Description	Form
0		Identifier	"$PNORB"
1		Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	n
2		Processing method (0, 1, 2, 3)	n
3		Frequency Low	d.dd
4		Frequency High	d.dd
5		Hm0 (m)	dd.dd
6		Tm02 (s)	dd.dd
7		Tp (s)	dd.dd
8		DirTp (deg)	dd.d
9		SprTp (deg)	dd.d
10		Main Direction (deg)	dd.d
11		Error Code	hh
12		Checksum (hex)	*hh

Example:
$PNORB,120704,130301,3,2,0.02,0.20,2.61,7.75,50.00,216.06,62.83,225.71,0*0
6

Chapter 7
Example Program

For your conveninence, we are pleased to provide a few example programs.

The following examples are provided:

• Generating a break
• Decoding the data structures – using Aquadopp as an example
• Structure definitions

Generating a Break

```
//////////////////////////////////////////////////////////////////////////////
// Sample code using the Microsoft Win32 API to open a handle to COM1,
// configure the serial port and send a break signal to wake up the instrument.

  .
  .
  .

  DCB dcb;
  HANDLE hComm;
  DWORD dwError;
  DWORD nBytesWritten;
  char cCommand[10];

  // Open a handle to COM1
  hComm = CreateFile("COM1",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);
  if (hComm == INVALID_HANDLE_VALUE) {
    dwError = GetLastError();
    // Handle the error.
  }
  // Omit the call to SetupComm to use the default queue sizes.
  // Get the current configuration.
  if (!GetCommState(hComm,&dcb)) {
    dwError = GetLastError();
```

```
    // Handle the error.
  }

  // Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit.
  dcb.BaudRate = 9600;
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;

  if (!SetCommState(hComm, &dcb)) {
    dwError = GetLastError();
    // Handle the error.
  }

  // Send a soft break signal
  memset(cCommand,64,6);     // @@@@@@
  if (!WriteFile(hComm,cCommand,6,&nBytesWritten,NULL))
    dwError = GetLastError();
    // Handle the error.
  }
  Sleep(100);
  strcpy(cCommand,"K1W%!Q");
  if (!WriteFile(hComm,cCommand,6,&nBytesWritten,NULL))
    dwError = GetLastError();
    // Handle the error.
  }

  // Send a hard break signal
  // Place the transmission line in a break state for 500 milliseconds
  SetCommBreak(hComm);
  Sleep(500);
  ClearCommBreak(hComm);

  .
  .
  .
```

Decoding the Data Structures

```
//////////////////////////////////////////////////////////////////////
// Sample code for decoding the Aquadopp data structure

typedef struct {
  unsigned char  cSync;         // sync = 0xa5
  unsigned char  cId;           // identification (0x01=normal, 0x80=diag)
  unsigned short hSize;         // size of structure (words)
  PdClock        clock;         // date and time
  short          hError;        // error code
  short          hSpare;
  unsigned short hBattery;      // battery voltage (0.1 V)
  unsigned short hSoundSpeed;   // speed of sound  (0.1 m/s)
  short          hHeading;      // compass heading (0.1 deg)
  short          hPitch;        // compass pitch   (0.1 deg)
```

```
  short          hRoll;        // compass roll    (0.1 deg)
  unsigned char  cMSB;         // pressure MSB
  char           cStatus;      // status code
  unsigned short hLSW;         // pressure LSW
  short          hTemperature; // temperature (0.01 deg C)
  short          hVel[3];      // velocity  (mm/s)
  unsigned       char cAmp[3]; // amplitude (counts)
  char           cFill;
  short          hChecksum;    // checksum
} PdMeas;


{
  .
  .
  .

  PdMeas meas;
  SYSTEMTIME st;
  double dVel[3];
  double dAmp[3];
  short hChecksum;
  double dPressure;
  double dBattery;
  double dHeading;
  double dPitch;
  double dRoll;
  double dTemperature;


  // Assuming three beams

  // Checksum control
  if (meas.hChecksum != Checksum((short *)&meas,meas.hSize - 1)) {
    // Handle the error.
  }

  st = ClockToSystemTime(meas.clock);

  dVel[0] = (double)meas.hVel[0] * 0.001;
  dVel[1] = (double)meas.hVel[1] * 0.001;
  dVel[2] = (double)meas.hVel[2] * 0.001;
  dAmp[0] = (double)meas.cAmp[0];
  dAmp[1] = (double)meas.cAmp[1];
  dAmp[2] = (double)meas.cAmp[2];

  dPressure = (65536.0*(double)meas.cMSB + (double)meas.hLSW)*0.001;
  dBattery = (double)meas.hBattery * 0.1;
  dHeading = (double)meas.hHeading * 0.1;
  dPitch = (double)meas.hPitch * 0.1;
  dRoll = (double)meas.hRoll * 0.1;
  dTemperature = (double)meas.hTemperature * 0.01;
  .
  .
  .
```

```
}//////////////////////////////////////////////////////////////////////
// Convert from BCD time to system time

SYSTEMTIME ClockToSystemTime(PdClock clock)
{
  SYSTEMTIME systime;
  WORD wYear;

  wYear = (WORD)BCDToChar(clock.cYear);
  if (wYear >= 90) {
    wYear += 1900;
  }
  else {
    wYear += 2000;
  }

  systime.wYear = wYear;
  systime.wMonth = (WORD)BCDToChar(clock.cMonth);
  systime.wDay = (WORD)BCDToChar(clock.cDay);
  systime.wHour = (WORD)BCDToChar(clock.cHour);
  systime.wMinute = (WORD)BCDToChar(clock.cMinute);
  systime.wSecond = (WORD)BCDToChar(clock.cSecond);
  systime.wMilliseconds = 0;

  return systime;
}


//////////////////////////////////////////////////////////////////////
// Convert from BCD to char

unsigned char BCDToChar(unsigned char cBCD)
{
  unsigned char c;

  cBCD = min(cBCD,0x99);
  c  = (cBCD & 0x0f);
  c += 10 * (cBCD >> 4);

  return c;
}


//////////////////////////////////////////////////////////////////////
// Compute checksum

short Checksum(short *phBuff,int n)
{
  int i;
  short hChecksum = 0xb58c;

  for (i=0; i<n; i++)
    hChecksum += phBuff[i];
```

```
  return hChecksum;
}
```

Structure Definitions

```
#define PD_MAX_BEAMS            3
#define PD_MAX_BINS           128
#define PD_MAX_STAGECELLS        1024


#pragma pack(push)
#pragma pack(1)     // 1 byte struct member alignment used in firmware


/////////////////////////////////////////////////////////////////////////////
// Clock data (6 bytes)  NOTE! BCD format

typedef struct {
   unsigned char  cMinute;       // minute
   unsigned char  cSecond;       // second
   unsigned char  cDay;          // day
   unsigned char  cHour;         // hour
   unsigned char  cYear;         // year
   unsigned char  cMonth;        // month
} PdClock;


/////////////////////////////////////////////////////////////////////////////
// Aquadopp diagnostics header data

typedef struct {
   unsigned char  cSync;         // sync = 0xa5
   unsigned char  cId;           // identification = 0x06
   unsigned short hSize;         // total size of structure (words)
   unsigned short nRecords;      // number of diagnostics samples to follow
   unsigned short nCell;         // cell number of stored diagnostics data
   unsigned char  cNoise[4];     // noise amplitude (counts)
   PdClock        clock;         // date and time
   unsigned short hSpare1;
   unsigned short hDistance[4];  // distance
   unsigned short hSpare[3];
   short          hChecksum;     // checksum
} PdDiagHead;


/////////////////////////////////////////////////////////////////////////////
// Aquadopp velocity data 3 beams

typedef struct {
   unsigned char  cSync;         // sync = 0xa5
   unsigned char  cId;           // identification (0x01=normal, 0x80=diag)
   unsigned short hSize;         // size of structure (words)
   PdClock        clock;         // date and time
   short          hError;        // error code:
                                 //    bit 0: compass (0=ok, 1=error)
                                 //    bit 1: measurement data (0=ok, 1=error)
                                 //    bit 2: sensor data (0=ok, 1=error)
```

```
                        //    bit 3: tag bit (0=ok, 1=error)
                        //    bit 4: flash (0=ok, 1=error)
                        //    bit 5:
                        //    bit 6: serial CT sensor read (0=ok, 1=error)
  unsigned short hAnaIn1;       // analog input 1
  unsigned short hBattery;      // battery voltage (0.1 V)
  union {
    unsigned short hSoundSpeed;   // speed of sound (0.1 m/s)
    unsigned short hAnaIn2;       // analog input 2
  } u;
  short          hHeading;      // compass heading (0.1 deg)
  short          hPitch;        // compass pitch (0.1 deg)
  short          hRoll;         // compass roll (0.1 deg)
  unsigned char  cPressureMSB;  // pressure MSB
  char           cStatus;       // status:
                        //    bit 0: orientation (0=up, 1=down)
                        //    bit 1: scaling (0=mm/s, 1=0.1mm/s)
                        //    bit 2: pitch (0=ok, 1=out of range)
                        //    bit 3: roll (0=ok, 1=out of range)
                        //    bit 4: wakeup state:
                        //    bit 5: (00=bad power, 01=break, 10=power applied, 11=RTC alarm)
                        //    bit 6: power level:
                        //    bit 7: (00=0(high), 01=1, 10=2, 11=3(low))
  unsigned short hPressureLSW;  // pressure LSW
  short          hTemperature;  // temperature (0.01 deg C)
  short          hVel[3];       // velocity
  unsigned char  cAmp[3];       // amplitude
  char           cFill;
  short          hChecksum;     // checksum
} PdMeas;

///////////////////////////////////////////////////////////////////////////////
// Vector velocity data header (18 bytes)

typedef struct {
  unsigned char  cSync;         // sync = 0xa5
  unsigned char  cId;           // identification = 0x12
  unsigned short hSize;         // total size of structure (words)
  PdClock        clock;         // date and time
  unsigned short nRecords;      // number of velocity samples to follow
  unsigned char  cNoise[4];     // noise amplitude (counts)
  unsigned char  cCorr[4];      // noise correlation
  unsigned short hSpare[10];    // spare values
  short          hChecksum;     // checksum
} PdVecHead;


///////////////////////////////////////////////////////////////////////////////
// Vector velocity data 3 beams

typedef struct {
  unsigned char  cSync;         // sync = 0xa5
  unsigned char  cId;           // identification = 0x10
  unsigned char  cAnaIn2LSB;    // analog input 2 LSB
  unsigned char  cCount;        // ensemble counter
```

```
   unsigned char  cPressureMSB;  // pressure MSB
   unsigned char  cAnaIn2MSB;    // analog input 2 MSB
   unsigned short hPressureLSW;  // pressure LSW
   unsigned short hAnaIn1;       // analog input 1 (fast)
   short          hVel[3];       // velocity
   unsigned char  cAmp[3];       // amplitude
   unsigned char  cCorr[3];      // correlation  (0-100)
   short          hChecksum;     // checksum
} PdVecVel;


/////////////////////////////////////////////////////////////////////////////
// Vector system data (28 bytes)

typedef struct {
   unsigned char  cSync;         // sync = 0xa5
   unsigned char  cId;           // identification = 0x11
   unsigned short hSize;         // size of structure (words)
   PdClock        clock;         // date and time
   unsigned short hBattery;      // battery voltage (0.1 V)
   unsigned short hSoundSpeed;   // speed of sound (0.1 m/s)
   short          hHeading;      // compass heading (0.1 deg)
   short          hPitch;        // compass pitch (0.1 deg)
   short          hRoll;         // compass roll (0.1 deg)
   short          hTemperature;  // temperature (0.01 deg C)
   char           cError;        // error code
   char           cStatus;       // status
   unsigned short hAnaIn;        // analog input (slow)
   short          hChecksum;
} PdVecSys;


/////////////////////////////////////////////////////////////////////////////
// Aquadopp velocity profile data

typedef struct {
   unsigned char  cSync;         // sync = 0xa5
   unsigned char  cId;           // identification (0x21 = 3 beams, 0x22 = 2 beams, 0x21 = 1 beam)
   unsigned short hSize;         // size of structure (words)
   PdClock        clock;         // date and time
   short          hError;        // error code
   unsigned short hAnaIn1;       // analog input 1
   unsigned short hBattery;      // battery voltage (0.1 V)
   union {
      unsigned short hSoundSpeed;// speed of sound (0.1 m/s)
      unsigned short hAnaIn2;    // analog input 2
   } u;
   short          hHeading;      // compass heading (0.1 deg)
   short          hPitch;        // compass pitch (0.1 deg)
   short          hRoll;         // compass roll (0.1 deg)
   union {
      struct {
         unsigned char  cMSB;    // pressure MSB
         char           cStatus; // status
         unsigned short hLSW;    // pressure LSW
```

```
    } Pressure;              // (mm)
    struct {
      unsigned char  cQuality;// distance quality
      char           cStatus; // status
      unsigned short hDist;   // distance (mm)
    } Distance;
  } u1;
  short          hTemperature;  // temperature (0.01 deg C)
  // actual size of the following = nBeams*nBins*3 + 2
  short          hVel[PD_MAX_BEAMS][PD_MAX_BINS]; // short hVel[nBeams][nCells];   // velocity
  unsigned char  cAmp[PD_MAX_BEAMS][PD_MAX_BINS]; // char  cAmp[nBeams][nCells];   // amplitude
                                                  // char  cFill                    // if nCells % 2 != 0
  short          hChecksum;     // checksum
} PdAqdProf;


/////////////////////////////////////////////////////////////////////////////////
// Continental velocity profile data (variable length)

typedef struct {
  unsigned char  cSync;         // sync = 0xa5
  unsigned char  cId;           // identification (0x24 = 3 beams, 0x25 = 2 beams, 0x26 = 1 beam)
  unsigned short hSize;         // size of structure (words)
  PdClock        clock;         // date and time
  short          hError;        // error code
  unsigned short hAnaIn1;       // analog input 1
  unsigned short hBattery;      // battery voltage (0.1 V)
  union {
    unsigned short hSoundSpeed;// speed of sound (0.1 m/s)
    unsigned short hAnaIn2;    // analog input 2
  } u;
  short          hHeading;      // compass heading (0.1 deg)
  short          hPitch;        // compass pitch (0.1 deg)
  short          hRoll;         // compass roll (0.1 deg)
  unsigned char  cPressureMSB;  // pressure MSB
  char           cStatus;       // status
  unsigned short hPressureLSW;  // pressure LSW
  short          hTemperature;  // temperature (0.01 deg C)
  short          hSpare[44];
  // actual size of the following = nBeams*nBins*3 + 2
  short          hVel[PD_MAX_BEAMS][PD_MAX_BINS]; // short hVel[nBeams][nCells];   // velocity
  unsigned char  cAmp[PD_MAX_BEAMS][PD_MAX_BINS]; // char  cAmp[nBeams][nCells];   // amplitude
                                                  // char  cFill                    // if nCells % 2 != 0
  short          hChecksum;     // checksum
} PdFarProf;


/////////////////////////////////////////////////////////////////////////////////
// AWAC velocity profile data (variable length)

typedef struct {
  unsigned char  cSync;         // sync = 0xa5
  unsigned char  cId;           // identification (0x20)
  unsigned short hSize;         // size of structure (words)
  PdClock        clock;         // date and time
```

```
  short       hError;      // error code
  unsigned short hAnaIn1;      // analog input 1
  unsigned short hBattery;     // battery voltage (0.1 V)
  union {
    unsigned short hSoundSpeed;  // speed of sound (0.1 m/s)
    unsigned short hAnaIn2;      // analog input 2
  } u;
  short       hHeading;     // compass heading (0.1 deg)
  short       hPitch;       // compass pitch (0.1 deg)
  short       hRoll;        // compass roll (0.1 deg)
  unsigned char  cPressureMSB;  // pressure MSB
  char        cStatus;      // status
  unsigned short hPressureLSW;  // pressure LSW
  short       hTemperature;  // temperature (0.01 deg C)
  short       hSpare[44];
  // actual size of the following = nBeams*nBins*3 + 2
  short       hVel[PD_MAX_BEAMS][PD_MAX_BINS]; // short hVel[nBeams][nCells];   // velocity
  unsigned char  cAmp[PD_MAX_BEAMS][PD_MAX_BINS]; // char  cAmp[nBeams][nCells];   // amplitude
                                      // char  cFill              // if nCells % 2 != 0
  short       hChecksum;    // checksum
} PdProf;


/////////////////////////////////////////////////////////////////////////////
// Wave header data (60 bytes)

typedef struct {
  unsigned char  cSync;       // sync = 0xa5
  unsigned char  cId;         // identification = 0x31
  unsigned short hSize;       // total size of structure (words)
  PdClock     clock;       // date and time
  unsigned short nRecords;     // number of wave data records to follow
  unsigned short hBlanking;    // T2 used for wave data measurements (counts)
  unsigned short hBattery;     // battery voltage (0.1 V)
  unsigned short hSoundSpeed;   // speed of sound (0.1 m/s)
  short       hHeading;     // compass heading (0.1 deg)
  short       hPitch;       // compass pitch (0.1 deg)
  short       hRoll;        // compass roll (0.1 deg)
  unsigned short hMinPress;    // minimum pressure value of previous profile (mm)
  unsigned short hMaxPress;    // maximum pressure value of previous profile (mm)
  short       hTemperature;  // temperature (0.01 deg C)
  unsigned short hCellSize;    // cell size in counts of T3
  unsigned char  cNoise[4];    // noise amplitude (counts)
  unsigned short hProcMagn[4];  // processing magnitude
  unsigned short hWindRed;     // number of samples of AST window past boundary
  unsigned short hASTWindow;    // AST window size (# samples)
  short       hSpare[5];    // spare values
  short       hChecksum;    // checksum
} PdWaveHead;


/////////////////////////////////////////////////////////////////////////////
// Wave data (24 bytes)

typedef struct {
```

```
    unsigned char  cSync;          // sync = 0xa5
    unsigned char  cId;            // identification (0x30)
    unsigned short hSize;          // size of structure (words)
    unsigned short hPressure;      // pressure (mm)
    unsigned short hDistance;      // AST distance1 on vertical beam (mm)
    unsigned short hAnaIn;         // analog input
    short          hVel[4];        // velocity, hVel[3] = AST distance2 on vertical beam (mm)
    unsigned char  cAmp[4];        // amplitude, cAmp[3] = AST quality (counts)
    short          hChecksum;      // checksum
} PdWave;
```