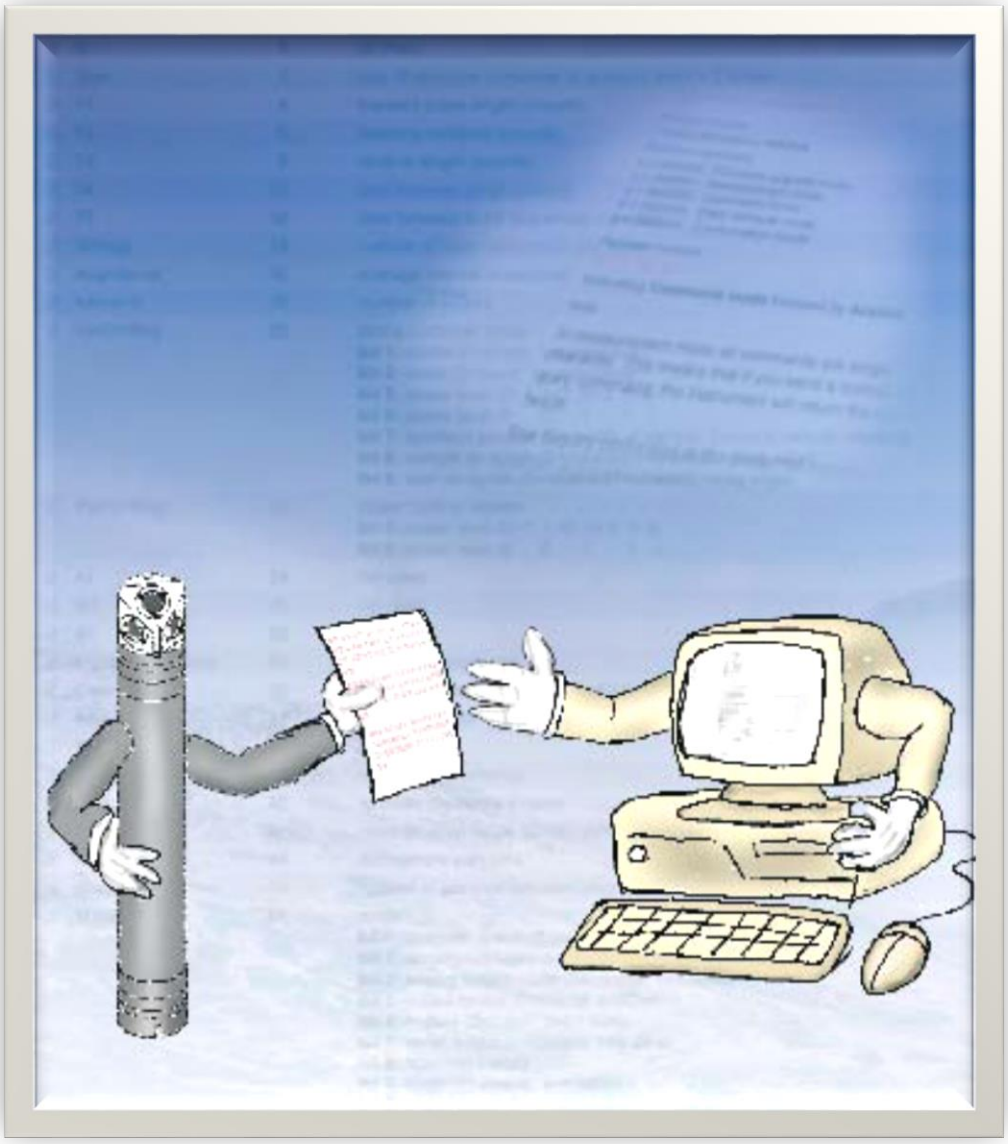
NORTEK AS

SYSTEM INTEGRATOR MANUAL



Dec 2014

SYSTEM INTEGRATOR MANUAL

Copyright © Nortek AS 2010. All rights reserved.

This document may not – in whole or in part – be copied, photocopied, translated, converted or reduced to any electronic medium or machine-readable form without prior consent in writing from Nortek AS. Every effort has been made to ensure the accuracy of this manual. However, Nortek AS makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability and fitness for a particular purpose. Nortek AS shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance or use of this manual or the examples herein. Nortek AS reserves the right to amend any of the information given in this manual in order to take account of new developments.

Microsoft, ActiveX, Windows, Windows 2000, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, service marks, or trade names of Nortek AS or other entities and may be registered in certain jurisdictions including internationally.

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

1 INTRODUCTION

This document provides the information needed to control a Nortek 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 Principles of Operation chapter in the Comprehensive Manual, available at the Nortek web.

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 write their own Windows applications to control one or more Nortek products an ActiveX® object is available. This greatly simplifies interfacing and the handling of the internal data structures.

Note that the Nortek 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 will not 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 check http://www.nortek-as.com/en/support or 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.

1.1 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 support 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.

2 BASIC INTERFACE CONCEPTS

The Nortek products communicate 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.

2.1 OPERATIONAL MODES

The operational modes for any Nortek 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.

For more details, see the Instrument States document available in Appendix A.

2.2 THE BREAK

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 Nortek system must be able to send a break.

To send a break you first send "@@@@@@" followed by a delay of 100 ms and then send "K1W%!Q".

2.3 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.

2.4 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, the Nortek instrument will discard both characters.

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.

2.5 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.

3 USE WITH A CONTROLLER

This chapter provides useful information when setting up your Nortek instrument with a controller. 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 behavior, 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, the Aquadopp Deep-water Current Meter, the AWAC and the Continental can output data in ASCII format – see the chapter on ASCII Output for more information. In addition the Aquadopp Profiler, the AWAC, the Aquadopp Current Meter, the Aquadopp Deep-water Current Meter are supporting the NMEA format.

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

3.1 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 will not 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.

3.2 CONTROL THE INSTRUMENT DIRECTLY

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 the controller to time-stamp the data, since Nortek instruments may lose 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.

3.3 Turning the Power On/Off

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 deployment planning 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.

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), all the data will be recorded to the same file. This allows easy verification of the controller since what has been logged to the internal recorder is identical to what has been sent to the controller.

When controlling a continuously measuring instrument with Prolog installed, make sure that the Prolog has enough time to flush the collected data before power is removed. The Prolog will flush data every 32kB when the instrument is configured to measure continuously. If the power is removed before 32kB of data is collected, no data is flushed.

3.4 Power Consumption

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.

3.5 Control via the Serial Line

In this case, the data collection is controlled over the serial line. To start data collection, a two-character ASCII command is sent; for instance AD. 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.

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	
		Measuring
....) (*&) (*&) (&		Outputs binary data
&!%#&)*J ASH(#&		
		Aquadopp is powered down

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

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 tables below.

Example: For the RC command 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

Please note than when using the terminal emulator embedded in Nortek software the little endian conversion is done by the terminal emulator.

BCD format: Binary coded decimal is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Four bits are used per digit. We show the binary sequences in hex.

The different instrument states and how to change state is illustrated in Appendix A: Instrument States

4.1 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 09 07 02 11 10 12 06 06 (2. December 2010 11:09:07)
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 = 0x3232 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 a high speed RS232/422 to USB converter.

Save baud rate	
Execute command **SB** **Hex** **4253**	**Description** Saves the currently set baud rate
	Response **AckAck**
	Parameter **0x4733 0x3241**
Reference Chapter on Data Structures	**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 user 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 **GH** **Hex** **4847**	**Description** Read the currently used head configuration from the instrument
	Response Head configuration (224 bytes) followed by **AckAck**

	Parameter None
	Response example a5 04 70 00 1f 40 d0 07 00 00 41 51 50 20 34 33 35 34 00 00 00 00 00 00 00 2d 00 00 00 50 0b . . . 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 03 00 f0 60 06 06
Reference Chapter on Data Structures	**Note** Some lines in the above example have been removed for clarity.

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

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	**Description**

AD **Hex** **4441**	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. Upon 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
	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.

Start ASCII measurement without recorder	
Execute command **AS** **Hex** **5341**	**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.

Acquire ASCII data	
Execute command **MA** **Hex** **414D**	**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. Upon receipt of a break it will jump directly to **Command Mode.**

Start NMEA measurement without recorder	
Execute command **NM** **Hex** **4D4E**	**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 **XXXX** or **0000**
	Parameter None
Reference	**Note** **XXXX** is the number of seconds before the first message is output (=average interval). **0000** is NMEA's NACK most likely due to too low baudrate

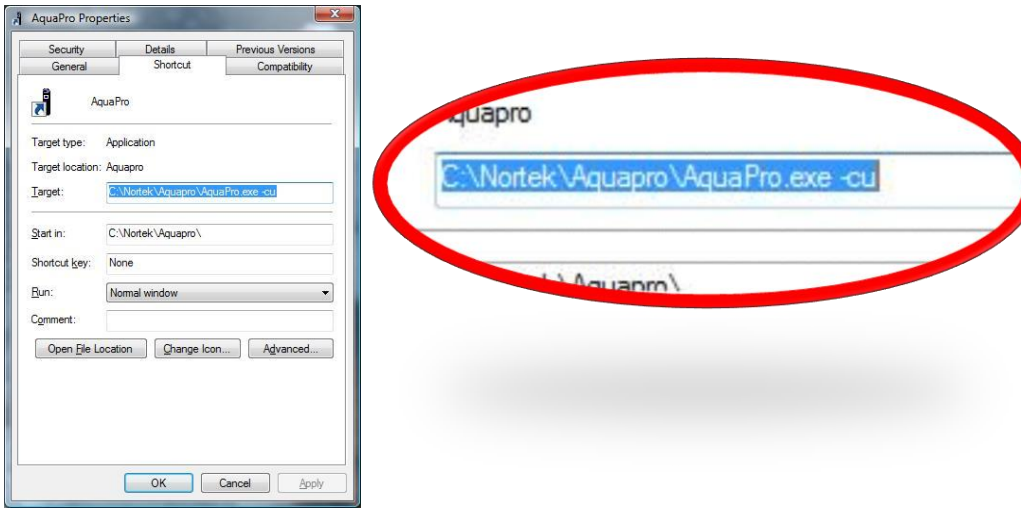
Acquire NMEA data	
Execute command **NS** **Hex** **534E**	**Description** Starts a single measurement based on the current configuration of the instrument without storing data to the recorder. Instrument enters **Power Down Mode** when measurement has been made.
	Response **XXXX** or **0000**
	Parameter None
Reference	**Note** **XXXX** is the number of seconds before the first message is output (=average interval). **0000** is NMEA's NACK most likely due to too low baudrate. 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. Upon receipt of a break, it will jump directly to **Command Mode.**

Configure instrument	
Execute command **CC** **Hex** **4343**	**Description** 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 is described below.

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 do as follows:

1. Generate a new shortcut to the Nortek software. Right click on the AquaPro (in this example) icon
2. Append the characters -cu in the target line as shown below (using AquaPro 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.
5. Open the .pcf file in a binary editor and insert 4343 at the very beginning of the file.
6. Save the file as .pdc
7. In the terminal emulator check the binary mode check box
8. Choose Command File and the .pdc file

4.2 Data Collection Mode

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. If the standard inquiry command is sent ('II'), the system will send the response twice.

These commands are not available when a 4GB logger/prolog is installed.

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.

4.3 In Confirmation Mode

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**

Wave Interval – ProLog only	
Execute command **WI** **Hex** **4957**	**Description** This command changes the wave interval. Takes a 32 bit argument and sets the parameter to the specified number of seconds
	Response **AckAck or NackNack**
	Parameter 32 bit argument
	Example *60 min interval = 3600 sec = Ox 0E 10* 57 49 00 00 10 0E
Reference	**Note** For newer versions of AWAC with ProLog. Can be used in Data retrieval mode and Command mode.

Measurement Interval	
Execute command **MI** **Hex** **494D**	**Description** This command changes the Measurement Interval. Takes a 16 bit argument and sets the parameter to the specified number of seconds
	Response **AckAck or NackNack**
	Parameter 16 bit argument
	Example *10 min interval = 600 sec = Ox 02 58* 4D 49 58 02
Reference	**Note** For newer versions of AWAC, AquaPro and Vector. Can be used in Data retrieval mode and Command mode.

Continue	
Execute command **CO** **Hex** **4F43**	**Description** The CO command can be used instead of the SR command to get the data and set the clock without going into Command Mode.
	Response **AckAck or NackNack**
	Parameter None
Reference	**Note** For newer versions of AWAC, AquaPro and Vector.

5 FIRMWARE DATA STRUCTURES

This section describes the data structures that are used for the Nortek 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-specific Structures
- Vector-specific Structures
- Aquadopp Profiler-specific Structures
- AWAC-specific Structures
- Continental

5.1 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 words 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 words 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 For Vector AvgInterval = 512/Sampling Rate
2	NBeams	18	number of beam

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 for most instruments For Vector it holds Number of Samples Per Burst
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 words 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 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 wraparound 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.

If the recorder is full, it can be formatted by use of the FO command. In Hex, the command to erase the recorder is 46 4F 12 D4 1E EF.

5.2 AQUADOPP AND AQUADOPP DW 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 (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)

1	Status	25	status code
2	PressureLSW	26	pressure LSW (0.001 dbar) (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 words 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 words in structure
Total size 36 Bytes			

Aquadopp Diagnostics Data

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

5.3 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	Spare	15	
1	Correlation1	16	noise correlation beam 1
1	Correlation2	17	noise correlation beam 2
1	Correlation3	18	noise correlation beam 3
1	Spare	19	
20	Spare	20	spare
2	Checksum	40	= b58c(hex) + sum of all words 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 (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)
1	AnaIn2MSB	5	analog input 2 MSB
2	PressureLSW	6	pressure LSW (0.001 dbar) (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 words 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 words in structure
Total Size 28 Bytes			

Vector and Vectrino Probe Check Data

The structure of the probe check is the same for both Vectrino and Vector. The difference is that a Vector has 3 beams and 300 samples, while the Vectrino has 4 beams and 500 samples.

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	07(hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Samples	4	Number of samples per beam
2	First sample	6	First sample number
1	Amp 1 B1	7	amplitude cell 1, beam1 (counts)
1	Amp 2…n	8	...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
2	Checksum		= b58c(hex) + sum of all words in structure
Total Size is variable			

Vector With IMU

The Vector may be outfitted with an optional Inertial Motion Unit. The sensor package is composed of a triaxial accelerometer, triaxial gyro, and a triaxial magnetometer. The obvious value of the sensor is that it not only provides accurate information about attitude and motion, but it estimates are sampled and recorded at the same rate as the Vector. This permits corrections for motion and orientation for Vectors that are subject to motion (e.g., buoy or mooring line configurations). Note that the gravitational constant uses fixed reference of g=9.80665 m/s^2.

The analysis required to correct and characterize motion is a non-trivial problem. It is assumed that the end user has competence in this area. It may benefit the end user to visit Microstrain's web site (www.microstrain.com) and navigate to the 3DM-GX3-25 product; here one can find supporting documentation on specifications, methods, and elemental technical notes.

If there is a need to make adjustments to the way IMU is integrated, it is possible to use a transparent mode making it possible to connect directly to the IMU sensor, and then make use of the Microstrain software (named 3DM-GX3 Monitor). The Hex command to send is 54 53 00 00. It will work on 9600 baud, but 115000 is preferable.

The structure of the IMU differs according to the selected deployment configuration:

Acceleration, Angular Rate, Magnetometer Vectors and Orientation Matrix:

Size	Name	Offset	Description
1	Sync	0	Sync = 0xa5
1	Id	1	Identification = 0x71
2	Size	2	Size of structure in number of words (1 word = 2 bytes)
1	EnsCnt	4	Ensemble Counter
1	AHRSId	5	AHRS ID 0xcc
4	Accel x	6	Acceleration x (m/s^2)
4	Accel y	10	Acceleration y (m/s^2)
4	Accel z	14	Acceleration z (m/s^2)
4	AngRt x	18	Angular Rate x (°/s)
4	AngRt y	22	Angular Rate y (°/s)
4	AngRt z	26	Angular Rate z (°/s)
4	MagRt x	30	Magnetometer Rate x (Gauss)
4	MagRt y	34	Magnetometer Rate y (Gauss)
4	MagRt z	38	Magnetometer Rate y (Gauss)
4	MatrixX	42	Orientation Matrix X (M_{11})
4	MatrixX	46	Orientation Matrix X (M_{12})
4	MatrixX	50	Orientation Matrix X (M_{13})
4	MatrixY	54	Orientation Matrix Y (M_{21})
4	MatrixY	58	Orientation Matrix Y (M_{22})
4	MatrixY	62	Orientation Matrix Y (M_{23})
4	MatrixZ	66	Orientation Matrix Z (M_{31})
4	MatrixZ	70	Orientation Matrix Z (M_{32})
4	MatrixZ	74	Orientation Matrix Z (M_{33})
4	timer	78	Timestamp (s)
2	AHRSchecksum	82	AHRS Checksum
2	Checksum	84	sum of all words in structure

Total Size 86 Bytes

Gyro-stabilized Acceleration, Angular Rate and Magnetometer Vectors:

Size	Name	Offset	Description
1	Sync	0	Sync = 0xa5
1	Id	1	Identification = 0x71
2	Size	2	Size of structure in number of words (1 word = 2 bytes)
1	EnsCnt	4	Ensemble counter
1	AHRSId	5	AHRS ID 0xd2
4	Accel x	6	Acceleration x (m/s^2)
4	Accel y	10	Acceleration y (m/s^2)
4	Accel z	14	Acceleration z (m/s^2)
4	AngRt x	18	Angular Rate x (°/s)
4	AngRt y	22	Angular Rate y (°/s)
4	AngRt z	26	Angular Rate z (°/s)
4	MagRt x	30	Magnetometer Rate x (Gauss)
4	MagRt y	34	Magnetometer Rate y (Gauss)
4	MagRt z	38	Magnetometer Rate z (Gauss)
4	timer	42	Timestamp (s)
2	AHRSchecksum	46	AHRS Checksum
2	Checksum	48	sum of all words in structure
Total Size 50 Bytes			

DeltaAngle, DeltaVelocity and Magnetometer Vectors:

Size	Name	Offset	Description
1	Sync	0	Sync = 0xa5
1	Id	1	Identification = 0x71
2	Size	2	Size of structure in number of words (1 word = 2 bytes)
1	EnsCnt	4	Ensemble Counter
1	AHRSId	5	AHRS ID, 0xd3
4	Angle	6	Delta Angle x (°)
4	Angle	10	Delta Angle y (°)
4	Angle	14	Delta Angle z (°)
4	Veloc	18	Delta Velocity Rate x (m/s)
4	Veloc	22	Delta Velocity Rate y (m/s)
4	Veloc	26	Delta Velocity Rate z (m/s)

4	MagVe	30	Delta Mag Vector Rate x (Gauss)
4	MagVe	34	Delta Mag Vector Rate y (Gauss)
4	MagVe	38	Delta Mag Vector Rate z (Gauss)
4	timer	42	Timestamp (s)
2	AHRSchecksum	46	AHRS Checksum
2	Checksum	48	sum of all words in structure
Total Size 50 Bytes			

5.4 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 (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	status code
2	PressureLSW	26	pressure LSW (0.001 dbar) (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 words in structure
Total Size is variable		

Aquadopp Profiler Wave Burst Data

Same as AWAC Wave Data and AWAC Wave Data Header

5.5 HR Aquadopp Specific Structures

High Resolution Aquadopp Profiler Data

Size	Name	Offset	Description
1	Sync	0	a5(hex)
1	Id	1	2a (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	Milliseconds	10	Milliseconds
2	Error	12	Error code
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°)
1	PressureMSB	24	Pressure MSB (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	Status code

2	PressureLSW	26	Pressure LSW (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)
2	Temperature	28	Temperature (0.01 °C)
2	AnaIn1	30	Analogue Input 1
2	AnaIn2	32	Analogue Input 2
1	Beams	34	Number of beams
1	Cells	35	Number of Cells
6	VelLag2[3]	36	Velocity Lag 2 - array of 3 - 1 per beam
3	AmpLag2[3]	42	Amplitude Lag 2 - array of 3 - 1 per beam
3	CorrLag2[3]	45	Correlation Lag 2 - array of 3 - 1 per beam
2	Spare 1	48	
2	Spare 2	50	
2	Spare 1	52	
	Vel[n Beams][n Cells]	54	Note that this part of the structure varies in size depending upon the number of beams and cells being used. Per beam and per cell Vel is 2 bytes, while Amp and Corr is 1 byte
	Amp[n Beams][n Cells]		
	Corr[n Beams][n Cells]		
2	hC Checksum		= b58c(hex) + sum of all words in structure
Total Size is variable			

Velocity, amplitude and correlation is output in the following order

Velocity Beam 1 cell 1
Velocity Beam 1 cell 2
Velocity Beam 1 cell 3
….
Velocity Beam 2 cell 1
Velocity Beam 2 cell 2
Velocity Beam 2 cell 3
….
Velocity Beam 3 cell 1
Velocity Beam 3 cell 2
Velocity Beam 3 cell 3
….
Amplitude Beam 1 cell 1
Amplitude Beam 1 cell 2
Amplitude Beam 1 cell 3
….
Amplitude Beam 2 cell 1
Amplitude Beam 2 cell 2
Amplitude Beam 2 cell 3
….
Amplitude Beam 3 cell 1
Amplitude Beam 3 cell 2
Amplitude Beam 3 cell 3
….
Correlation Beam 1 cell 1

Correlation Beam 1 cell 2
Correlation Beam 1 cell 3
….
Correlation Beam 2 cell 1
Correlation Beam 2 cell 2
Correlation Beam 2 cell 3
….
Correlation Beam 3 cell 1
Correlation Beam 3 cell 2
Correlation Beam 3 cell 3

5.6 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 (0.001 dbar) (Pressure = 65536×PressureMSB + PressureLSW)
1	Status	25	status code
2	PressureLSW	26	pressure LSW (0.001 dbar) (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 words 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 (0.1V)
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 (0.001 dbar)
2	MaxPress	26	max pressure value of previous profile (0.001 dbar)
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 words in structure
Total Size is 60 Bytes			

Awac Stage Data

Size	Name	Offset	Description
1	Sync	0	a5(hex)
1	Id	1	42 (hex)
2	Size	2	size of structure in number of words (1 word = 2 bytes)
2	Spare	4	(AST distance1 duplicate)
1	Amplitude1	6	amplitude beam 1 (counts)
1	Amplitude2	7	amplitude beam 2 (counts)
1	Amplitude3	8	amplitude beam 3 (counts)
1	Spare	9	(AST quality duplicate)
2	Pressure	10	pressure (0.001 dbar)
2	AST1	12	AST distance 1 (mm)
2	AST Quality	14	AST quality (counts)
2	SoundSpeed	16	Speed of sound (0.1 m/s)
2	AST2	18	AST distance 2 (mm)
2	Spare	20	
2	Velocity1	22	velocity beam 1 (mm/s) (East for SUV)
2	Velocity2	24	velocity beam 2 (mm/s) (North for SUV)
2	Velocity3	26	velocity beam 3 (mm/s) (Up for SUV)
2	Spare	28	(AST distance2 duplicate)
2	Spare	30	
1	Amp 1	32	amplitude cell 1 (counts)
1	Amp 2..n	33	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 words in structure
Total Size is Variable			

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 (0.001 dbar)
2	Distance1	6	distance 1 to surface vertical beam (mm)
2	AnaIn	8	analog input
2	Vel1	10	velocity beam 1 (mm/s) (East for SUV)
2	Vel2	12	velocity beam 2 (mm/s) (North for SUV)
2	Vel3	14	velocity beam 3 (mm/s) (Up for SUV)
2	Distance 2/ Vel4	16	distance 2 to surface vertical beam (mm) For non-AST 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	AST quality Counts) For non-AST amplitude beam 4 (mm/s)
2	Checksum	22	= b58c(hex) + sum of all words in structure
Total Size is 24 Bytes			

Awac Wave Data for SUV

Size	Name	Offset	Description
1	Sync	0	a5(hex)
1	Id	1	36 (hex)
2	Heading	2	heading (0.1deg)
2	Pressure	4	pressure (0.001 dbar)
2	Distance	6	distance 1 to surface vertical beam (mm)
1	Pitch	8	pitch (0.1 or 0.2 deg) (+/- 12.7 deg)
1	Roll	9	roll (0.1 or 0.2 deg) (+/- 12.7 deg)
2	Vel1	10	velocity beam 1 (mm/s) (East for SUV)
2	Vel2	12	velocity beam 2 (mm/s) (North for SUV)
2	Vel3	14	velocity beam 3 (mm/s) (Up for SUV)
2	Distance 2/ Vel4	16	distance 2 to surface vertical beam (mm) For non-AST 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	AST quality Counts For non-AST amplitude beam 4 (mm/s)
2	Checksum	22	= b58c(hex) + sum of all words in structure
Total Size is 24 Bytes			

5.7 CONTINENTAL DATA

Same as AWAC Profiler Data, except ID = 0x24

5.8 PROLOG

The Prolog is a module that can be added to Nortek instruments in replacement of the standard recorder. It is installed in the location as the static 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 is in line with the Nortek philosophy that the data should still be available even in the event of damage to the instrument that leads to a leakage.

The Prolog can have added wave processing functionality when used with an AWAC. Data may then be streamed over 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 the instrument deployment software (e.g., AWAC AST software).

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 basis for comma separated data. Section 6.3 NMEA Output is a description of how the different data are formatted.

Prolog specific structures

The recorded processed, binary data is composed of the instrument's 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	Sync	0	A5 (hex)
1	ID	1	60 (hex)
2	Size	2	size in words
6	clock	4	date and time
1	hSpectrumTyp	10	spectrum used for calculation
1	hProcMethod	11	processing method used in actual calculation
2	Hm0	12	Spectral significant wave height [mm]
2	H3	14	AST significant wave height (mean of largest 1/3) [mm]
2	H10	16	AST wave height(mean of largest 1/10) [mm]
2	Hmax	18	AST max wave height in wave ensemble [mm]
2	Tm02	20	Mean period spectrum based [0.01 sec]
2	Tp	22	Peak period [0.01 sec]
2	Tz	24	AST mean zero-crossing period [0.01 sec]
2	DirTp	26	Direction at Tp [0.01 deg]
2	SprTp	28	Spreading at Tp [0.01 deg]
2	DirMean	30	Mean wave direction [0.01 deg]
2	UI	32	Unidirectivity index [1/65535]
4	hPressureMean	34	Mean pressure during burst [0.001 dbar]
2	NumNoDet	38	Number of AST No detects

2	NumBadDet	40	Number of AST Bad detects
2	CurSpeedMean	42	Mean current speed - wave cells [mm/sec]
2	CurDirMean	44	Mean current direction - wave cells [0.01 deg]
4	hError	46	Error Code for bad data
4	ASTdistMean	50	Mean AST distance during burst [mm]
4	ICEdistMean	54	Mean ICE distance during burst [mm]
2	FreqDirAmbLimit	58	Low frequency in [0.001 Hz]
2	T3	60	AST significant wave period (sec)
2	T10	62	AST 1/10 wave period (sec)
2	Tmax	64	AST max period in wave ensemble (sec)
2	Hmean	66	Mean wave height (mm)
10	Spares	68	
2	Checksum	78	checksum
Total Size is 80 Bytes			

Wave band estimates

Size	Name	Offset	Description
1	Sync	0	A5 (hex)
1	ID	1	61 (hex)
2	Size	2	size in words
6	Clock	4	date and time
1	SpectrumType	10	spectrum used for calculation
1	ProcMethod	11	processing method used in actual calculation
2	LowFrequency	12	low frequency in [0.001 Hz]
2	HighFrequency	14	high frequency in [0.001 Hz]
2	Hm0	16	Spectral significant wave height [mm]
2	Tm02	18	Mean period spectrum based [0.01 sec]
2	Tp	20	Peak period [0.01 sec]
2	DirTp	22	Direction at Tp [0.01 deg]
2	DirMean	24	Mean wave direction [0.01 deg]
2	SprTp	26	Spreading at Tp [0.01 deg]
4	Error	28	Error Code for bad data
14	Spares	32	
2	Checksum	46	checksum
Total Size is 48 Bytes			

Wave energy spectrum

Size	Name	Offset	Description
1	Sync	0	A5(hex)
1	ID	1	62 (hex)
2	Size	2	size in words
6	Clock	4	date and time
1	SpectrumType	10	spectrum used for calculation
1	Spare	11	
2	NumSpectrum	12	Number of spectral bins (default 98)
2	LowFrequency	14	low frequency in [0.001 Hz]
2	HighFrequency	16	high frequency in [0.001 Hz]
2	StepFrequency	18	frequency step in [0.001 Hz]
18	Spares	20	
4	EnergyMultiplier	38	AST energy spectrum multiplier [cm^2/Hz]
var	Energy	42	AST Spectra [0 - 1/65535] -
2	Checksum	.	checksum
Total Size is Variable			

Wave fourier coefficient spectrum

Size	Name	Offset	Description
1	cSync	0	A5 (hex)
1	cID	1	63 (hex)
2	Size	2	size in words
6	clock	4	date and time
1	cSpare	10	
1	cProcMethod	11	processing method used in actual calculation
2	NumSpectrum	12	Number of spectral bins (default 49)
2	LowFrequency	14	low frequency in [0.001 Hz]
2	HighFrequency	16	high frequency in [0.001 Hz]
2	StepFrequency	18	frequency step in [0.001 Hz]
10	Spares	20	
196	A1	30	Fourier coefficients in [+/- 1/32767]
196	B1	226	Fourier coefficients in [+/- 1/32767]
196	A2	422	Fourier coefficients in [+/- 1/32767]
196	B2	618	Fourier coefficients in [+/- 1/32767]
2	Checksum	814	checksum
Total Size is 816 Bytes			

Awac Cleaned Up AST Time Series

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	65 (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	Samples	10	Number of wave samples (AST samples = 2*nSamples)
12	Spare	12	
2	AST 2*Samples		AST distance time series (mm)
2	Checksum		= b58c(hex) + sum of all words in structure
Total size is variable, depending on number of samples			

Data that 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.

Awac Processed Velocity Profile Data

Size	Name	Offset	Description
1	Sync	0	a5 (hex)
1	Id	1	6a (hex)
2	Size	2	Size of structure in number of words (1word = 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	Milliseconds	10	Milliseconds
1	Beams	12	Number of beams
1	Cells	13	Number of cells
2	Vel 1 (B1/X/E)	14	Velocity cell 1, beam1/X/East
2	Vel 2 … n	16…	…repeated for cells 2 through n
2	Vel 1 (B2/Y/N)		Velocity cell 1, beam2/Y/North
2	Vel 2 … n		…repeated for cells 2 through n
2	Vel 1 (B3/Z/U)		Velocity cell 1, beam2/Y/North
2	Vel 2 … n		…repeated for cells 2 through n
2	SNR 1 B1		Signal-to-Noise ratio, cell 1, beam 1 (counts)

2	SNR 2 … n	…repeated for cells 2 through n
2	SNR 1 B2	Signal-to-Noise ratio, cell 1, beam 2 (counts)
2	SNR 2 … n	…repeated for cells 2 through n
2	SNR 1 B3	Signal-to-Noise ratio, cell 1, beam 3 (counts)
2	SNR 2 … n	…repeated for cells 2 through n
2	STD 1 B1	Standard deviation cell 1, beam 1
2	STD 2 … n	…repeated for cells 2 through n
2	STD 1 B2	Standard deviation cell 1, beam 2
2	STD 2 … n	…repeated for cells 2 through n
2	STD 1 B3	Standard deviation cell 1, beam 3
2	STD 2 … n	…repeated for cells 2 through n
1	CellErr 1, B1	Error code, cell 1, beam 1
1	CellErr 2… n	…repeated for cells 2 through n
1	CellErr 1, B2	Error code, cell 1, beam 2
1	CellErr 2… n	…repeated for cells 2 through n
1	CellErr 1, B3	Error code, cell 1, beam 3
1	CellErr 2… n	…repeated for cells 2 through n
2	Speed 1…n	Speed
2	Dir 1…n	Direction
2	VertDist 1..n	Vertical distance (m)
1	ProfErr 1…n	Error code
1	QCflag 1…n	Flag. 0= not eval; 1 = bad; 2 = questionable; 3 = good
2	Checksum	= b58c(hex) + sum of all words in structure
Total size is variable, depending in number of samples		

5.9 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 words 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
			e = error (0 = no error, 1 = error condition)
			x = not used
			v = velocity scaling (0 = mm/s, 1 = 0.1mm/s)
			ccc = #cells -1
			bb = #beams -1

1	Count	3	ensemble counter (0 - 255)
2	Vel 1 B1/X	4	velocity cell 1, beam1 or X (mm/s)
2	Vel 1 B2/Y	6	velocity cell 1, beam2 or Y (mm/s)
2	Vel 1 B3/Z	8	velocity cell 1, beam3 or Z (mm/s)
2	Vel 1 B4/Z2	10	velocity cell 1, beam4 or Z2 (mm/s)
1	Amp 1 B1	12	amplitude cell 1, beam1 (counts)
1	Amp 1 B2	13	amplitude cell 1, beam2 (counts)
1	Amp 1 B3	14	amplitude cell 1, beam3 (counts)

1	Amp 1 B4	15	amplitude cell 1, beam4 (counts)
1	Corr 1 B1	16	correlation cell 1, beam1 (%)
1	Corr 1 B2	17	correlation cell 1, beam2 (%)
1	Corr 1 B3	18	correlation cell 1, beam3 (%)
1	Corr 1 B4	19	correlation cell 1, beam4 (%)
2	Checksum	20	= b58c(hex) + sum of all words in structure
Total Size is 22 Bytes			

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 words in structure
Total Size is 16 Bytes			

5.10 Error And Status Codes

Error

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aquadopp	**Coord. Transf** 0=ok 1=error	**CT sensor** 0=ok 1=error	**Beam number** 0=ok 1=error	**Flash** 0=ok 1=error	**Tag bit** 0=ok 1=error	**Sensor data** 0=ok 1=error	**Measurement data** 0=ok 1=error	**Compass** 0=ok 1=error
Vector	**Coord. Transf** 0=ok 1=error		**Beam number** 0=ok 1=error		**Tag bit** 0=ok 1=error	**Sensor data** 0=ok 1=error	**Measurement data** 0=ok 1=error	**Compass** 0=ok 1=error
AquaPro HR	**Coord. Transf** 0=ok 1=error		**Beam number** 0=ok 1=error	**Flash** 0=ok 1=error	**Tag bit** 0=ok 1=error	**Sensor data** 0=ok 1=error	**Measurement data** 0=ok 1=error	**Compass** 0=ok 1=error
Other	**Coord. Transf** 0=ok 1=error		**Beam number** 0=ok 1=error	**Flash** 0=ok 1=error	**Tag bit** 0=ok 1=error	**Sensor data** 0=ok 1=error	**Measurement data** 0=ok 1=error	**Compass** 0=ok 1=error

Status

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aquadopp DW	**Power level** 00=0 (high)		**Wakeup state** 00=bad power 01=power applied 10=break 11=RTC alarm		**Roll** 0=ok 1= out of range	**Pitch** 0=ok 1= out of range	**Scaling** 0=mm/s 1=0.1mm/s	**Orientation** 0=up 1=down
Vector	**Power level** 00=0 (high) 10=2(low)		**Wakeup state** 00=bad power 01=power applied 10=break 11=RTC alarm		**Roll** 0=ok 1= out of range	**Pitch** 0=ok 1= out of range	**Scaling** 0=mm/s 1=0.1mm/s	**Orientation** 0=up 1=down
Other	**Power level** 00=0 (high) 01=1 10=2 11=3 (low)		**Wakeup state** 00=bad power 01=power applied 10=break 11=RTC alarm		**Roll** 0=ok 1= out of range	**Pitch** 0=ok 1= out of range	**Scaling** 0=mm/s 1=0.1mm/s	**Orientation** 0=up 1=down

Vector users: Note Bit 1 of the Status code. This bit indicates the scaling of the velocity output and depends on the velocity range setting. If the instrument is set to use the highest ranges, the least significant bit is 1 mm/s. For the lowest range, it is 0.1 mm/s. The purpose of varying scale factor is to make sure we utilize as much as we can of the dynamic range that is inherent in the system. This is all transparent if you use the Vector software to convert to ASCII because the data reported in the ASCII files is in engineering units. If you develop your own program to read the binary data files, the variable scaling needs to be taken into account.

6 ASCII OUTPUT

Most Nortek instruments can output ASCII data directly. There are three ways of doing this.

- Disk recording in ASCII format
- ASCII output from MA/AS commands
- Prolog ASCII serial output

6.1 DISK RECORDING IN ASCII FORMAT

The output format is the same as the standard output of the ASCII conversion using the Nortek software. 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 for instance an .aqd data file to ASCII.

6.2 ASCII OUTPUT FROM MA/AS COMMANDS

Some Nortek instrument are also capable of sending out ASCII formatted data.

There are two 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.

The following should be observed:

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 Nortek software by entering the required setup parameters and updating the instrument with this deployment planning.

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.

To start a measurement with output in ASCII format the following steps must be used:

1. Set the relevant deployment parameters using the Nortek software that is shipped with the instrument.

2. Download the deployment configuration to the instrument by using the update function in the deployment planning.

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 Nortek SW. Alternatively, the measurement can be stopped by sending a soft break followed by the characters MC (Mode Command). Note that there is no storage of data to the internal recorder when data are output in ASCII format.

The parameters are variable in size, but space delimited.

6.2.1 AQUADOPP PROFILER AND AWAC (NON-AST)

The format is as follows:

Header Line

Name	Units
Month	(1-12)
Day	(1-31)
Year	
Hour	(0-23)
Minute	(0-59)
Second	(0-59)
Error Code	
Status Code	
Battery voltage	V
Soundspeed	m/s
Heading	degrees
Pitch	degrees
Roll	degrees
Pressure	dbar
Temperature	degrees C
Analogue input 1	Counts (0 - 65536)
Analogue input 2	Counts (0 - 65536)

Data Line

Name	Units
CellNo	(1-128)
Speed	mm/s
Direction	tenth of degrees

Here is an example with one data set with three cells

11 11 2010 16 25 11 129 33 13.6 1429.9 128.3 0.3 -0.4 1.113 -6.98 6204 6205
1 1289 3102
2 544 393
3 1178 2852

6.2.2 CONTINENTAL

The format is as follows:

Header Line

Name	Units
Serial No.	
Year	
Month	(1-12)
Day	(1-31)
Hour	(0-23)
Minute	(0-59)
Second	(0-59)
Temperature	degrees C
Spare	

Data Line

Name	Units
CellNo	(1-128)
Speed	mm/s
Direction	tenth of degrees

Here is an example with one data set with three cells

CNH5689 2010 12 1 10 26 46 2328 0

1 2163 2305

2 2847 3577

3 4884 3244

6.2.3 AQUADOPP

The format is as follows:

Name	Units
Month	(1-12)
Day	(1-31)
Year	
Hour	(0-23)
Minute	(0-59)
Second	(0-59)
Error Code	
Status Code	
Velocity (Beam1/X/East)	m/s
Velocity (Beam2/Y/North)	m/s
Velocity (Beam3/Z/Up)	m/s
Amplitude (Beam1)	counts
Amplitude (Beam2)	counts
Amplitude (Beam3)	counts
Battery	volt
Soundspeed	m/s
Heading	degrees
Pitch	degrees
Roll	degrees
Pressure	dbar
Temperature	degrees C
Analogue input 1	Counts (0 - 65536)
Analogue input 2	Counts (0 - 65536)
Speed	m/s
Direction	Degrees

Example:

12 1 2010 11 21 47 0 160 -0.600 1.147 0.496 29 26 36 13.2 1531.8 140.0 -0.3 -13.4 108.422 23.93 0 0 1.294 332.4

6.3 NMEA OUTPUT

Aquadopp, Aquadopp DW, Aquadopp Profiler and the AWAC are currently supporting the NMEA format.

The diagnostic samples for the Aquadopp are output in the same way as an other measurement.

For the AWAC this is configured in the deployment planning while for the rest it must be started by the NM/NS commands.

Data with variants of -9 (-9.00, -999…) are invalid data.

Empty files are fields not used.

$PNORC,073010,050000,1,0.10,-0.11,-0.01,0.15,137.2,C,88,83,87,,,*37

Correlation is not used for the AWAC

The checksum calculation is part of the NMEA standard. It is the representation of two hexadecimal characters of an XOR if all characters in the sentence between – but not including – the $ and the * character.

Information (configuration)

Field	Description	Form
0	Identifier	"$PNORI"
1	Instrument type	0=Aquadopp, 2= Aquadopp Profiler 3=AWAC
2	Head ID	aaannnn
3	Number of beams	N
4	Number of cells	N
5	Blanking (m)	dd.dd
6	Cell size (m)	dd.dd
7	Coordinate system	ENU=0, XYZ=1,Beam=2
8	Checksum	*hh

Example:

$PNORI,3,WAV6103,3,20,0.51,2.00,0*16

Sensor Data

Field	Description	Form
0	Identifier	"$PNORS"
1	Date	MMDDYY
2	Time	hhmmss
3	Error code (hex)	hh
4	Status code (hex)	hh
5	Battery voltage (V)	dd.d
6	Sound speed (m/s)	dddd.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)	nnnnn
13	Analog input #2 (counts)	nnnnn
14	Checksum (hex)	*hh

Example:

$PNORS,073010,050000,00,B0,13.4,1520.6,114.9,-0.5,1.6,22.314,18.92,1039,0*0B

Current velocity data

Field	Description	Form
0	Identifier	"$PNORC"
1	Date	MMDDYY
2	Time	hhmmss
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:

$PNORC,073010,050000,1,0.10,-0.11,-0.01,0.15,137.2,C,88,83,87,,,*37

$PNORC,073010,050000,2,0.15,-0.16,-0.02,0.22,138.1,C,76,71,74,,,*3D

Wave parameters

Field	Description	Form
0	Identifier	"$PNORW"
1	Date	MMDDYY
2	Time	hhmmss
3	Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	n
4	Processing method (1-PUV, 2-SUV, 3-MLM, 4-MLMST)	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)	ddd.dd
13	SprTp (deg)	ddd.dd
14	Main Direction (deg)	ddd.dd
15	Unidirectivity Index	dd.dd
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)	ddd.dd
21	Error Code	hhhh
22	Checksum (hex)	*hh

Example:

$PNORW,073010,051001,3,4,0.55,0.51,0.63,0.82,2.76,3.33,2.97,55.06,78.91,337.62,0.48,22.35,0,1,0.27,129.11,0000*4E

Wave energy density spectrum

Field	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	nnn
7	Energy Density [frequency 1] (cm2/Hz)	dddd.ddd
8	Energy Density [frequency 2] (cm2/Hz)	dddd.ddd
N+6	Energy Density [frequency N] (cm2/Hz)	dddd.ddd
N+7	Checksum (hex)	*hh

Example:

$PNORE,073010,051001,3,0.02,0.01,98,0.000,0.000,0.000,0.001,0.001,0.001,0.001,0.001,0.001,0.001,0.001,0.001,0.002,0.002,0.002,0.002,0.002,0.002,0.003,0.003,0.004,0.006,0.010,0.023,0.049,0.091,0.162,0.176,0.213,0.179,0.160,0.104,0.097,0.072,0.056,0.036,0.032,0.034,0.040,0.032,0.028,0.021,0.017,0.017,0.014,0.012,0.009,0.011,0.010,0.012,0.009,0.010,0.009,0.007,0.006,0.007,0.007,0.008,0.007,0.006,0.005,0.004,0.004,0.003,0.003,0.003,0.003,0.002,0.003,0.003,0.002,0.002,0.002,0.002,0.002,0.001*7E

Wave band parameters

Field	Description	Form
0	Identifier	"$PNORB"
1	Date	MMDDYY
2	Time	hhmmss.s
3	Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	n
4	Processing method (1-PUV, 2-SUV, 3-MLM, 4-MLMST)	n
5	Frequency Low	d.dd
6	Frequency High	d.dd
7	Hm0 (m)	dd.dd
8	Tm02 (s)	dd.dd
9	Tp (s)	dd.dd
10	DirTp (deg)	ddd.dd
11	SprTp (deg)	ddd.dd

11	Main Direction (deg)	ddd.dd
12	Error Code	hhhh
13	Checksum (hex)	*hh

Example:

$PNORB,073010,051001,3,4,0.02,0.20,0.06,7.06,5.00,262.39,80.27,23.39,0000*62

$PNORB,073010,051001,3,4,0.21,0.49,0.52,3.06,3.33,57.06,78.91,24.66,0000*50

Fourier coefficient spectra

Field	Description	Form
0	Identifier	"$PNORF"
1	Fourier coefficient flag [A1/B1/A2/B2]	"CC"
2	Date	MMDDYY
3	Time	hhmmss.s
4	Spectrum basis type (0-pressure, 1-Velocity, 3-AST)	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,A1,073010,051001,3,0.02,0.01,48,-0.0216,-0.0521,-0.0563,-0.0565,-0.0287,-0.0149,-0.0099,-0.0531,-0.0445,-0.0431,-0.0204,-0.0141,0.0697,0.0833,0.0540,0.0190,-0.0195,-0.0367,-0.0025,-0.0143,0.0318,-0.0307,-0.0051,0.0041,0.0440,0.0114,0.0831,0.0527,0.0284,0.0104,0.0040,0.0030,0.0049,-0.0005,0.0001,-0.0007,0.0018,0.0011,0.0012,0.0008,0.0029,0.0035,0.0021,-9.0000,-9.0000,-9.0000,-9.0000,-9.0000*0B

7 MAKING A NORTEK FILE – EXAMPLE .VEC

When data is collected in an integrated system it is not necessary readable by our software. Our different software is reading files as they are stored and downloaded by our software setup software (as Vector).

The requirement for a complete Nortek file is a the following content

- Hardware configuration (A5 05)
- Head configuration (A5 04)
- User configuration (A5 00)
- Measurement data

All three configurations may be made or retrieved from the instrument by the GA command. Each of the configurations may be made or retrieved by GP for the hardware configuration, GH for the head configuration and GC for the user configuration.

Example .VEC file

A complete Vector file contains

- Hardware configuration (A5 05)
- Head configuration (A5 04)
- User configuration (A5 00)
- Probe check data (A5 07)
- Vector Velocity data header (A5 10)
- Vector System Data (A5 11)
- Probe check data (A5 07)

8 OVERVIEW OF THE IDS

A5 00	User Configuration
A5 01	Aquadopp Velocity Data
A5 02	Vectrino distance data
A5 04	Head Configuration
A5 05	Hardware Configuration
A5 06	Aquadopp Diagnostics Data Header
A5 07	Vector and Vectrino Probe Check data
A5 10	Vector Velocity Data
A5 11	Vector System Data
A5 12	Vector Velocity Data Header
A5 20	AWAC Velocity Profile Data
A5 21	Aquadopp Profiler Velocity Data
A5 24	Continental Data
A5 2a	High Resolution Aquadopp Profiler Data
A5 30	AWAC Wave Data
A5 31	AWAC Wave Data Header
A5 36	AWAC Wave Data SUV
A5 42	AWAC Stage Data
A5 50	Vectrino velocity data header
A5 51	Vectrino velocity data
A5 60	Wave parameter estimates
A5 61	Wave band estimates
A5 62	Wave energy spectrum
A5 63	Wave Fourier coefficient spectrum
A5 65	Cleaned up AST time series
A5 6a	Awac Processed Velocity Profile Data
A5 80	Aquadopp Diagnostics Data

9 INDUCTIVE MODEM INTEGRATION

The following options are available for deployment planning in the Aquadopp software:

• enabling the IMM

• setting the device ID in the IMM

• setting the transmit power level

• selecting ASCII or binary format

The parameters are not set in the modem until the deployment is started. During the deployment process the IMM configuration is stored in the deployment log file by the Aquadopp software, and the complete configuration of the Aquadopp is stored in the Host File in the IMM. This enables the surface inductive modem (SIM/IMM) to retrieve the Aquadopp configuration through the command HostFileGetData. These data are only stored in binary format, so if this command is used the SIM must be configured for binary data. This is also the case if Aquadopp binary format is selected for storage in the IMM. The following commands are the most relevant for use in the SIM for retrieving data from the Aquadopp:

• !iiSampleGetList

• !iiSampleGetData:

• !iiSampleGetLast

• !iiSampleEraseAll

• !iiHostFileGetData

The file example.log on the following page is an example SIM session for binary data transfer. A corresponding example for ASCII data is shown in the file data01.log. The corresponding converted Aquadopp file data01.dat shows the converted data from the internal recorder in the Aquadopp. –

example.log

IMM>captureline
<Executing/>
!15HostFileGetData
<RemoteReply><Executing/>
<HostData Len='784' CRC='0x43FFA2E2'>

Binary data returned:

```
                  a5 05 18 20 41   51 44 20 35 35 30 35 20
20 20 20 20 20 0c 20 7a   22 20 20 03 20 90 20 2c
20 20 20 41 fa 20 20 7a   22 41 fa 20 20 39 69 33
33 40 ce a5 04 70 20 0f   20 d0 07 20 20 41 51 44
20 30 36 39 38 20 20 20   20 20 20 20 20 20 20 20
20 50 0b 50 0b 20 20 b0   f4 50 0b 20 20 b0 f4 b0
f4 a0 16 01 80 a8 20 c0   34 b3 fe 01 80 77 06 39
34 f7 fe 20 20 20 20 20   20 ff ff 20 20 01 20 20
20 01 20 20 20 20 20 20   20 20 20 ff ff 20 20 ff
ff 20 20 ff ff 20 20 20   20 20 20 ff ff 01 20 20
20 20 20 ff ff ff ff 20   20 20 20 01 20 20 10 20
20 c9 05 03 01 d8 1b 6d   2a 01 80 13 01 68 2e 38
20 01 80 c9 02 69 2e 78   ff ff 7f d9 01 53 01 d9
01 af 7e e0 f7 7a fc c9   fb ae 73 67 20 e0 ff 88
ff 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 55 15 10 0e 10 0e 10   27 64 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 03
20 7e a4 a5 20 20 01 7d   20 31 20 20 20 b5 01 20
02 01 20 3c 20 03 20 40   20 20 20 20 20 20 20 20
20 02 20 20 20 01 20 20   20 2c 01 69 6d 6d 74 73
74 20 20 30 20 07 12 09   04 10 0e 20 20 02 20 11
41 14 20 01 20 14 20 04   06 21 20 2e 33 5e 01 02
3d 1e 3d 39 3d 53 3d 6e   3d 88 3d a2 3d bb 3d d4
3d ed 3d 06 3e 1e 3e 36   3e 4e 3e 65 3e 7d 3e 93
3e aa 3e c0 3e d6 3e ec   3e 02 3f 17 3f 2c 3f 41
3f 55 3f 69 3f 7d 3f 91   3f a4 3f b8 3f ca 3f dd
3f f0 3f 02 40 14 40 26   40 37 40 49 40 5a 40 6b
40 7c 40 8c 40 9c 40 ac   40 bc 40 cc 40 db 40 ea
40 f9 40 08 41 17 41 25   41 33 41 42 41 4f 41 5d
41 6a 41 78 41 85 41 92   41 9e 41 ab 41 b7 41 c3
41 cf 41 db 41 e7 41 f2   41 fd 41 08 42 13 42 1e
42 28 42 33 42 3d 42 47   42 51 42 5b 42 64 42 6e
42 77 42 80 42 89 42 91   42 9a 42 a2 42 aa 42 b2
42 ba 42 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 2a
20 0f 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 06 20 03 20 20 20 20   20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
20 0a ff cd ff 8b 20 e5   20 ee 20 0b 20 84 ff 3d
ff 4b 5a
```

</HostData>
<Executed/>
</RemoteReply>
<Executed/>
IMM>releaseline
<Executing/>
<Executed/>
IMM>
4 <TIMEOUT msg='HostService 2 min timeout'/>
IMM>
IMM>
<Executed/>
IMM>captureline
<Executing/>
<Executed/>
IMM>!15samplegetsummary
<RemoteReply><Executing/>
<SampleDataSummary NumSamples='8' TotalLen='336' FreeMem='16006'/>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!15samplegetlist
<RemoteReply><Executing/>
<SampleList>
<Sample ID='0x0000024c' Len='42' CRC='0xC7589EDD'/>
<Sample ID='0x0000024b' Len='42' CRC='0x4C593F18'/>
<Sample ID='0x0000024a' Len='42' CRC='0x457FCB04'/>
<Sample ID='0x00000249' Len='42' CRC='0xD6C60EE2'/>
<Sample ID='0x00000248' Len='42' CRC='0x8AFE5C1A'/>
<Sample ID='0x00000247' Len='42' CRC='0x2301B4DB'/>
<Sample ID='0x00000246' Len='42' CRC='0xB3B4B2D4'/>
<Sample ID='0x00000245' Len='42' CRC='0x0BCBBDE2'/>
</SampleList>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:245
<RemoteReply><Executing/>
<SampleData ID='0x245' LEN='42' CRC='0xbcbbde2'>

Binary data returned:

```
                  a5 01 15   20 35 20 07 12 09 04 20
20 ff ff 7d 20 20 20 33   0d ed ff e1 ff 20 b1 b3
37 48 fd cc ff 8e 04 23   03 0e 0e 0e 20 9c d6
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:246
<RemoteReply><Executing/>
<SampleData ID='0x246' LEN='42' CRC='0xb3b4b2d4'>

Binary data returned:

```
                                                    a5 01
15 20 40 20 07 12 09 04  20 20 ff ff 7d 20 20 20
3c 0d ed ff e1 ff 20 b1  92 38 48 fd 16 fd 02 05
3f 03 0e 0e 0e 20 69 d5
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:247
<RemoteReply><Executing/>
<SampleData ID='0x247' LEN='42' CRC='0x2301b4db'>

Binary data returned:

```
                  a5 01  15 20 45 20 07 12 09 04
20 20 ff ff 7d 20 20 20  33 0d ed ff e1 ff 20 b1
99 38 48 fd e0 ff 74 06  bb 06 0e 0e 0e 20 24 dd
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:248
<RemoteReply><Executing/>
<SampleData ID='0x248' LEN='42' CRC='0x8afe5c1a'>

Binary data returned:

```
                                                    a5 01
15 20 50 20 07 12 09 04  20 20 ff ff 7d 20 20 20
37 0d ed ff e1 ff 20 b1  4b 38 48 fd 3a fd 7b 05
d8 04 0e 0e 0e 20 63 d7
```

<Executed/>
<Executed/>
IMM>!15SAMPLEGETDATA:249
<RemoteReply><Executing/>
<SampleData ID='0x249' LEN='42' CRC='0xd6c60ee2'>

Binary data returned:

```
                  a5 01  15 20 55 20 07 12 09 04
20 20 ff ff 7d 20 20 20  33 0d ed ff e1 ff 20 b1
7f 39 48 fd 9c fe 18 05  76 09 0e 0e 0e 20 35 de
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:24A
<RemoteReply><Executing/>
<SampleData ID='0x24a' LEN='42' CRC='0x457fcb04'>

Binary data returned:

```
a5 01 15 20 20 20 07 13   09 04 20 20 ff ff 7d 20
20 20 33 0d ed ff e1 ff   20 b1 ab 38 48 fd 29 01
22 03 ca 04 0e 0e 0e 20   f7 d9
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEGETDATA:24B
<RemoteReply><Executing/>
<SampleData ID='0x24b' LEN='42' CRC='0x4c593f18'>

Binary data returned:

```
                          a5 01 15 20 05 20 07 13
09 04 20 20 ff ff 7d 20   20 20 33 0d ed ff e1 ff
20 b1 81 38 48 fd 79 ff   33 07 bd 04 0e 0e 0e 20
26 dc
```

<Executed/>
</RemoteReply>
6 <Executed/>
IMM>!15SAMPLEGETDATA:24C
<RemoteReply><Executing/>
<SampleData ID='0x24c' LEN='42' CRC='0xc7589edd'>

Binary data returned:

```
a5 01 15 20 10 20 07 13   09 04 20 20 ff ff 7d 20
20 20 3c 0d ed ff e1 ff   20 b1 14 38 48 fd 7e fd
0d 01 b3 05 0e 0e 0e 20   a2 d4
```

<Executed/>
</RemoteReply>
<Executed/>
IMM>!15SAMPLEERASEALL
<RemoteReply><Executing/>
<Executed/>
</RemoteReply>
<Executed/>
IMM>releaseline
<Executing/>
<Executed/>
IMM>

data01.log

IMM>
<Executed/>
IMM>CaptureLine
<Executing/>
<Executed/>
IMM>!12SampleGetSummary
<RemoteReply><Executing/>
<SampleDataSummary NumSamples='5' TotalLen='550' FreeMem='15792'/>
<Executed/>

</RemoteReply>
<Executed/>
IMM>!12SampleGetList
<RemoteReply><Executing/>
<SampleList>
<Sample ID='0x00000254' Len='109' CRC='0x45326FC3'/>
<Sample ID='0x00000253' Len='110' CRC='0x74E6ED95'/>
<Sample ID='0x00000252' Len='109' CRC='0x0D7F6C29'/>
<Sample ID='0x00000251' Len='111' CRC='0x9175F508'/>
<Sample ID='0x00000250' Len='111' CRC='0x42E806B9'/>
</SampleList>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!12SAMPLEGETDATA:250
<RemoteReply><Executing/>
<SampleData ID='0x250' LEN='111' CRC='0x42e806b9'> 4 7 2009 13 35 0 0 177 -0.708 0.799 0.154 14 14 14 12.5 0.0 338.3 -1.9 -3.1 14.403 -6.96 65535 0 1.067 318.5
</SampleData>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!12SAMPLEGETDATA:251
<RemoteReply><Executing/>
<SampleData ID='0x251' LEN='111' CRC='0x9175f508'> 4 7 2009 13 45 0 0 177 -0.744 1.434 0.975 14 14 14 12.5 0.0 338.3 -1.9 -3.1 14.412 -6.96 65535 0 1.615 332.6
</SampleData>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!12SAMPLEGETDATA:252
<RemoteReply><Executing/>
<SampleData ID='0x252' LEN='109' CRC='0xd7f6c29'> 4 7 2009 13 55 0 0 177 0.292 0.669 0.810 14 14 14 12.5 0.0 338.3 -1.9 -3.1 14.371 -6.96 65535 0 0.730 23.6
</SampleData>
<Executed/>
</RemoteReply>
8 <Executed/>
IMM>!12SAMPLEGETDATA:253
<RemoteReply><Executing/>
<SampleData ID='0x253' LEN='110' CRC='0x74e6ed95'> 4 7 2009 14 5 0 0 177 -0.732 0.830 0.805 14 14 14 12.5 0.0 338.3 -1.9 -3.1 14.603 -6.96 65535 0 1.107 318.6
</SampleData>
<Executed/>
</RemoteReply>
<Executed/>
IMM>!12SAMPLEGETDATA:254
<RemoteReply><Executing/>
<SampleData ID='0x254' LEN='109' CRC='0x45326fc3'> 4 7 2009 14 15 0 0 177 0.343 0.964 0.796 14 14 14 12.5 0.0 337.9 -1.9 -3.1 14.493 -6.96 65535 0 1.023 19.6
</SampleData>
<Executed/>

</RemoteReply>
<Executed/>
IMM>!12SampleEraseAll
<RemoteReply><Executing/>
<Executed/>
</RemoteReply>
<Executed/>
IMM>releaseline
<Executing/>
<Executed/>

data01.dat

04 07 2009 13 35 00 00000000 10110001 -0.708 0.799 0.154 14 14 14 12.5
1414.6 338.3 -1.9 -3.1 14.403 -6.96 65535 0 1.068 318.46

04 07 2009 13 45 00 00000000 10110001 -0.744 1.434 0.975 14 14 14 12.5
1414.6 338.3 -1.9 -3.1 14.412 -6.96 65535 0 1.616 332.58

04 07 2009 13 55 00 00000000 10110001 0.292 0.669 0.810 14 14 14 12.5

1414.6 338.3 -1.9 -3.1 14.371 -6.96 65535 0 0.730 23.58

04 07 2009 14 05 00 00000000 10110001 -0.732 0.830 0.805 14 14 14 12.5
1414.6 338.3 -1.9 -3.1 14.603 -6.96 65535 0 1.107 318.59

04 07 2009 14 15 00 00000000 10110001 0.343 0.964 0.796 14 14 14 12.5
1414.6 337.9 -1.9 -3.1 14.493 -6.96 65535 0 1.023 19.59

10 EXAMPLE PROGRAM

For your convenience, 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

10.1 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);
.
.
.
```

10.2 DECODING OF 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;
}
```

10.3 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 (0.001 dbar)
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          c     lock; // 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; // (0.001 dbar)
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 c                      Status; // 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 (dbar)
unsigned short   hMaxPress; // maximum pressure value of previous profile (dbar)
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                    Spare[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 (0.001 dbar)
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;

////////////////////////////////////////////////////////////////////////////
// Wave Parameter Data (80 bytes)
typedef struct {
unsigned char    cSync; // A5 (hex)
unsigned char    cId; // 60 (hex)
unsigned short   hSize; // size in words
PdClock                clock; // date and time
unsigned char    cSpectrumType; // spectrum used for calculation
unsigned char    cProcMethod; // processing method used in actual calculation
unsigned short   hHm0; // Spectral significant wave height [mm]
unsigned short   hH3; // AST significant wave height (mean of largest 1/3) [mm]
unsigned short   hH10; // AST wave height(mean of largest 1/10) [mm]
unsigned short   hHmax; // AST max wave height in wave ensemble [mm]
unsigned short   hTm02; // Mean period spectrum based [0.01 sec]
unsigned short   hTp; // Peak period [0.01 sec]
unsigned short   hTz; // AST mean zero-crossing period [0.01 sec]
unsigned short   hDirTp; // Direction at Tp [0.01 deg]
unsigned short   hSprTp; // Spreading at Tp [0.01 deg]
unsigned short   hDirMean; // Mean wave direction [0.01 deg]
unsigned short   hUI; // Unidirectivity index [1/65535]
```

```
long                            lPressureMean; // Mean pressure during burst
                [0.001 dbar]
unsigned short   hNumNoDet; // Number of ST No detects [#]
unsigned short   hNumBadDet; // Number of ST Bad detects [#]
unsigned short   hCurSpeedMean; // Mean current speed - wave cells [mm/sec]
unsigned short   hCurDirMean; // Mean current direction - wave cells [0.01 deg]
unsigned long    lError; // Error Code for bad data
unsigned short   hSpares[14];
unsigned short   hChecksum; // checksum
} PdWaveParData


////////////////////////////////////////////////////////////////////////////
// Wave Band data (48 bytes)
typedef struct {
unsigned char    cSync; // A5 (hex)
unsigned char    cId; // 61 (hex)
unsigned short   hSize; // size in words
PdClock                clock; // date and time
unsigned char    cSpectrumType; // spectrum used for calculation
unsigned char    cProcMethod; // processing method used in actual calculation
unsigned short   hLowFrequency; // low frequency in [0.001 Hz]
unsigned short   hHighFrequency; // high frequency in [0.001 Hz]
unsigned short   hHm0; // Spectral significant wave height [mm]
unsigned short   hTm02; // Mean period spectrum based [0.01 sec]
unsigned short   hTp; // Peak period [0.01 sec]
unsigned short   hDirTp; // Direction at Tp [0.01 deg]
unsigned short   hDirMean; // Mean wave direction [0.01 deg]
unsigned short   hSprTp; // Spreading at Tp [0.01 deg]
unsigned long    lError; // Error Code for bad data
unsigned short   hSpares[7];
unsigned short   hChecksum; // checksum
} PdWaveBandsData;

////////////////////////////////////////////////////////////////////////////
// Wave Spectrum data (Variable size)
typedef struct {
unsigned char    cSync; // A5 (hex)
unsigned char    cId; // 62 (hex)
unsigned short   hSize; // size in words
PdClock                clock; // date and time
unsigned char    cSpectrumType; // spectrum used for calculation
unsigned char    cSpare;
unsigned short   hNumSpectrum; // number of spectral bins (default 98)
unsigned short   hLowFrequency; // low frequency in [0.001 Hz]
unsigned short   hHighFrequency; // high frequency in [0.001 Hz]
unsigned short   hStepFrequency; // frequency step in [0.001 Hz]
unsigned short   hSpares[9];
unsigned long    lEnergyMultiplier; // AST energy spectrum multiplier [cm^2/Hz]
unsigned short   hEnergy[PD _ MAX _ WAVEFREQST];
// AST Spectra [0 - 1/65535] -
unsigned short   hChecksum; // checksum
} PdWaveSpectrumData; // variable size (hNumSpectrum)


////////////////////////////////////////////////////////////////////////////
// Wave Fourier Coefficients (Variable size)
typedef struct {
unsigned char    cSync; // A5 (hex)
unsigned char    cId; // 63 (hex)
unsigned short    hSize; // size in words
```

```
PdClock                clock; // date and time
unsigned char    cSpare;
unsigned char    cProcMethod; // processing method used in actual calculation
unsigned short   hNumSpectrum; // number of spectral bins (default 49)
unsigned short   hLowFrequency; // low frequency in [0.001 Hz]
unsigned short   hHighFrequency; // high frequency in [0.001 Hz]
unsigned short   hStepFrequency; // frequency step in [0.001 Hz]
unsigned short   hSpares[5];
short                          hA1[PD _ MAX _ WAVEFREQ];
// Fourier coefficients in [+/- 1/32767]
short                          hB1[PD _ MAX _ WAVEFREQ];
// 0 - hNumSpectrum-1
short                          hA2[PD _ MAX _ WAVEFREQ];
short                          hB2[PD _ MAX _ WAVEFREQ];
unsigned short   hChecksum; // checksum
} PdWaveFourierCoeff;
```

11 APPENDIX A: INSTRUMENT STATES