SENSUS ULTRA

Developer's Guide

Last Updated: June 11, 2006

1.0 Introduction

This document describes the hardware/memory architecture and communication protocol of ReefNet's **Sensus Ultra** dive data recorder. Sensus Ultra was designed with third-party software developers in mind: the protocol is simple, efficient, and easy to implement.

1.1 The Sensus Product Family

Sensus Ultra is ReefNet's third-generation dive data recording system. Although the original Sensus and Sensus Pro are no longer in production, thousands are still in use today. Third-party developers are encouraged to support all three models, or at least the Pro and Ultra.

Separate developer's guides are available for each product generation. They may be downloaded for free from:

```
http://reefnet.ca/downloads/
```

1.2 News and Updates

To be notified about changes to this document, software updates, and new products, you should subscribe to ReefNet's mailing list:

```
http://reefnet.ca/newsletter/
```

1.3 Developer Loan Program

ReefNet understands that some developers cannot afford to purchase our hardware merely for testing purposes. If you fall into this category, then you might qualify for the *Developer Loan Program*: a limited number of hardware kits are available on loan for up to 2 months at a time.

To request a developer loan kit, send an email to **info@reefnet.ca** describing your project and how the loan will help you complete it. If your request is approved, you will be responsible only for shipping costs.

1.4 Technical Assistance

If you need technical help or discover an error in this document, please contact *Kris Wilk* at ReefNet:

Email: wilk@reefnet.ca

Phone: 1-888-819-7333 (US/Canada)
905-608-9373 (International)

Fax: 905-820-1927

Web: www.reefnet.ca

Mail: ReefNet Inc.
3610 Walnut Grove
Mississauga, Ontario L5L2W8
Canada

Before contacting ReefNet, test ReefNet's *Sensus Manager* software to verify that your hardware and connections are working. Then gather as much information about your problem as possible: symptoms, sample data, errors, unexpected results, etc.

2.0 Background

This chapter outlines important naming conventions, data types, and other background information required to understand subsequent chapters. Read carefully!

2.1 Basic Terminology

All references to the *device* refer to a Sensus Ultra recorder connected via the download unit to a *host* computer system. The *host* may be any of the following: PC, Mac, Palm handheld, Pocket PC, Unix workstation, etc.

2.2 Data Types

Only three basic data types are used during data transmissions. All are **unsigned integer** types, varying only in length:

Data Type	Description
UInt8	8-bit unsigned integer (byte)
UInt16	16-bit unsigned integer (word)
UInt32	32-bit unsigned integer (dword)

2.3 Byte Order

All data transmissions assume the **Little Endian (Intel)** byte significance convention. That is, the least significant byte(s) of a data variable are transmitted/stored first.

Be particularly careful on hosts which natively use the Big Endian (Motorola) convention, such as G4/G5 based Mac OS machines. You may need to swap bytes before or after transmission to ensure that data is not misinterpreted!

2.4 Error Detection

When sending or receiving data over a serial interface, it is not uncommon for one or more bits in the data stream to be corrupted due to electrical interference or a poor electrical connection. In order to make these bit errors detectable, the device implements an industry-standard **Cyclic Redundancy Check (CRC)** algorithm. The parameters used are consistent with the CRC-CCITT specification:

Length	16 bits
Polynomial	0x1021
Initial Value	0xFFFF
Reflection	none
Final XOR	none
Check String	“ReefNet” ---> 0xEF03

If you are not familiar with CRC error detection, an excellent tutorial is Ross Williams' "*A Painless Guide to CRC Error Detection Algorithms*", which can be found at:

```
http://www.ross.net/crc/crcpaper.html
```

Another good starting point is:

```
http://en.wikipedia.org/wiki/Cyclic_redundancy_check
```

Although you are encouraged to build a good understanding of CRC algorithms, you needn't write your own from scratch. Sample CRC routines can be found online for just about every programming language.

Whether you write a routine yourself or use someone else's code, ALWAYS test the Check String (above) to verify that the correct CRC is generated.

2.5 Encryption

A unique feature of the Sensus Ultra is its programmable **user segment**. This segment of memory can be used to store arbitrary user data of any type, such as certification information, an equipment list, a spreadsheet, etc.

ReefNet's *Sensus Manager* software includes a simple implementation of this feature: the user is permitted to upload a single data file into the segment. Before being uploaded, the user file is encrypted with the industry-standard **RC4 stream cipher**, initialized with a user-selected password.

If you are not familiar with RC4 encryption, a good starting point is:

```
http://en.wikipedia.org/wiki/Rc4
```

If you want to support ReefNet's implementation of the user segment, you will need to find or write a set of RC4 encryption routines and follow the guidelines in Section 4.1.

2.6 Timekeeping

Like a watch, the device contains an accurate clock that "ticks" **once every second**. Unlike a watch, this clock cannot be "set" to a particular date/time: it is simply a **32-bit counter** which starts at zero when the device is first assembled. When a dive occurs, the current clock value is stamped into the dive data.

To be useful, the timestamp must be translated into a real date/time. Actual dive times are computed by comparing the device clock with the host clock and "counting backwards". For example, consider a download with the following (convenient) characteristics:

```
Time of download (host clock)          2000-01-01 15:00
Time of download (device clock)        4600

Timestamp of a dive (device clock)     1000
```

These data imply that the dive started exactly 4600 - 1000 = 3600 seconds = **1 hour before** the download. Hence the actual dive start time is **2000-01-01 14:00**.

This system of timekeeping might seem awkward at first, but it's actually quite elegant. The host is the ultimate time reference, so the device clock never needs to be adjusted. The only requirement is that the host clock be set correctly.

2.7 Units of Measure

Voltage, temperature and pressure values are reported by the device as unsigned 16-bit integers (UInt16). Data are encoded as follows:

VOLTAGE

Battery voltage is reported in **millivolts [mV]**. For example, the value of 3456 indicates a voltage of 3.456 V.

The device contains a high-capacity primary lithium cell with a nominal voltage of 3.6 V. Note that remaining battery capacity **cannot be estimated** from the reported voltage value because the discharge curve for lithium cells is non-linear. In fact, the voltage will remain nearly constant until the battery is almost exhausted.

TEMPERATURE

Temperatures are reported in units of **0.01 Kelvin [K]**. For example, the value of 29815 indicates a temperature of 25.00 C. In case you've forgotten your high-school chemistry, 273.15 K = 0 C

PRESSURE

The device does not report depth. It reports the **absolute pressure in millibar [mbar abs]**. For example, the value 1000 indicates an absolute pressure of 1000 mbar -- a typical value near sea level.

Depth cannot be computed without first knowing (or assuming) the value of the absolute pressure at the sea surface, AND the water type (salt/fresh). Rather than make any assumptions, the device simply reports the absolute pressure and allows you to interpret the results according to your needs.

2.8 Recording Parameters

The overall behaviour of the device is governed by four **recording parameters**. Each is stored internally as a UInt16 value, and all four can be reprogrammed easily to suit the user's particular needs.

The recording parameters are:

INTERVAL (default = 10, valid range = 1 to 65535)

The sampling interval in **seconds**. Remember that the device capacity is proportional to INTERVAL. The advertised capacity of 1500 hours is based on the default interval of 10 seconds. If INTERVAL were reduced to 1, the capacity would be reduced to 150 hours.

THRESHOLD (default = 1111, valid range = 1 to 65535)

The activation pressure in **millibar absolute**. When this pressure is exceeded, the device enters dive mode and begins sampling pressure/temperature at the rate prescribed by INTERVAL. The default value of 1111 mbar corresponds to a depth of 3 fsw at sea level.

ENDCOUNT (default = 15, valid range = 1 to 65535)

The dive end count in **samples**. This value specifies the number of **consecutive** data samples *shallower than THRESHOLD* which constitute the end of a dive. With the default settings, a dive is terminated once the absolute pressure remains below 1111 mbar for at least 10 x 15 = 150 seconds.

AVERAGING (default = 1, valid range = 1, 2, or 4)

The device's pressure sensor is extremely sensitive -- its resolution is 1 mbar (about 1 cm of sea water!). But due to natural electrical noise, each raw pressure reading may be off by +/- 5 mbar. This can be remedied by averaging a few "rapid-fire" readings rather than taking just one.

The AVERAGING value specifies how many raw pressure readings are averaged **per data sample**. By default, no averaging is done. This is more than adequate for recreational and technical diving applications.

For scientific applications where very stable pressure measurements are required, AVERAGING can be set to either **2** or **4**. For instance, with 4x averaging, each data sample is actually the average of four pressure readings taken in quick succession.

Averaging mitigates the effect of noise at the expense of power. 2x and 4x averaging require 2 and 4 times as many raw pressure readings, respectively. Use only if absolutely necessary!

3.0 Hardware Architecture

The device communicates with the host via a simple 3-wire serial connection. This chapter describes the specific hardware requirements and necessary port settings.

3.1 Physical Connection

The device is designed to connect to any standard 9-pin (DB9) RS-232 serial port.

The download unit does not contain any electronics. It simply exposes the three serial port connections **RX**, **TX**, and **GND** to the device. The download unit's PC/Mac and Handheld ports differ as follows:

- The **PC/Mac port** is meant to connect to a **DTE** (*Data Terminal Equipment*) such as a PC, Mac, Unix workstation, or any other conventional desktop host offering a male DB9 serial port.
- The **Handheld port** is meant to connect to a **DCE** (*Data Communication Equipment*) such as a Palm handheld's cradle, or any other host offering a female DB9 serial port.

The two ports are wired together with the RX/TX lines swapped between them.

Only connect ONE host at a time -- connecting two hosts simultaneously (e.g., PC and Palm) may damage the hosts and/or the device.

Some newer hosts no longer offer serial ports. Fortunately, they can still be used with the device if a suitable adapter is purchased:

- **PCs or Macs with USB only** require a USB-serial adapter (such as ReefNet part # S-USB)
- **Handhelds/PDAs** require a serial cradle/cable or serial add-on card (options vary with PDA model)

Contact ReefNet if you need help connecting the device to the host.

3.2 Port Configuration

All communication with the device is done at **115.2 kbps**, with **8 data bits**, **no parity**, **1 stop bit**, and **NO flow control**.

4.0 Memory Architecture

The device contains a **2 MB** non-volatile flash memory. This memory is divided into two **segments**: the **USER** segment and the **DATA** segment. Each segment is subdivided into a number of **512 byte pages**.

4.1 The USER Segment

The **USER** segment is **32 pages** in length (16384 bytes). This segment is **readable and writeable** by the host and can be used to store data of any type. There is **no required structure** for this segment: you may use it however you prefer. You will learn how to read/write the USER segment in a later chapter.

ReefNet's particular implementation is described here in case you want to offer compatibility with the *Sensus Manager* software.

Sensus Manager allows the user to encrypt and store a single file in the USER segment. The stored file can be retrieved (with original file name intact) on any other host running *Sensus Manager*. This is achieved using the following **unencrypted** segment structure:

Offset	Type/Size	Name/Value
0x0000	UInt32	**0x00000000**
0x0004	UInt16	SIZE
0x0006	506 bytes	NAME
0x0200	up to 15872 bytes	FILE

The importance of the header value **0x00000000** will become clear later. The rest of the sections are:

SIZE

The size of the stored data file, in **bytes**. The maximum file size that can be stored with this implementation is **15872 bytes**.

NAME

A **null-terminated** string representing the name of the data file. For example, "myfile.txt**\n**".

FILE

The binary data of the file itself. SIZE is used to identify the end of the file, and unused space at the end of the USER segment is ignored.

After constructing the USER segment, *Sensus Manager* prompts the user for a **password**. The password is used to **initialize the RC4 encryption algorithm** (see Section 2.5), and the USER segment is encrypted. Finally, the encrypted USER segment is uploaded to the device.

The null byte (0x00) is always appended to the password. This ensures that even if no password is entered, the RC4 engine is initialized to a known state (initialized with "\n").

This encryption scheme is extremely strong because the password is stored neither on the host nor on the device. In fact, if the user forgets the password, the user data is virtually impossible to recover.

VERIFYING DECRYPTION

The RC4 algorithm is symmetric, meaning that the same process is used for both encryption and decryption. When the USER segment is read by *Sensus Manager*, the user is prompted for the password. The password (plus trailing null byte!) is used to initialize the RC4 algorithm and "decrypt" the retrieved data.

But remember that the software has no knowledge of the "correct" password. If an incorrect password is entered, the resulting USER segment data will simply be gibberish.

To distinguish between valid and invalid decryption results, the software **inspects the first 4 bytes** of the decrypted segment. When decrypted successfully, they should equal the UInt32 value **0x00000000**. Anything else indicates that the password must have been incorrect and the rest of the data are garbage.

4.2 The DATA Segment

The **DATA** segment is **4064 pages** in length (2080768 bytes). This segment is **read-only** and contains zero or more **dive records** in *chronological order*.

A **dive record** is a contiguous block of data which fully describes one dive. It is composed of three parts: a **header** which denotes the start of the record, a **body** which contains pairs of temperature and pressure values, and a **footer** that denotes the end of the record.

More specifically, the structure looks like:

	Offset	Type	Name/Value
HEADER	0x0000	UInt32	**0x00000000**
	0x0004	UInt32	TIMESTAMP
	0x0008	UInt16	INTERVAL
	0x000A	UInt16	THRESHOLD
	0x000C	UInt16	ENDCOUNT
	0x000E	UInt16	AVERAGING
BODY	0x0010	UInt16	TEMPERATURE
	0x0012	UInt16	PRESSURE
	0x0014	UInt16	TEMPERATURE
	0x0016	UInt16	PRESSURE
	. . .	. . .	. . .
	. . .	. . .	. . .
FOOTER	0x*NNNN*	UInt32	**0xFFFFFFFF**

Every dive record begins with the sequence **0x00000000** and ends with the sequence **0xFFFFFFFF**. Dive records can be located within the DATA segment by looking for the header start sequence.

Also be aware that the "erased" state of flash memory is **0xFF**, not 0x00.

The first dive record in the DATA segment does NOT necessarily begin at the start of the segment. Similarly, the last dive record does NOT necessarily end at the end of the segment. Ignore any "garbage" data found at either end…this data is normal and is a consequence of the internal memory buffering scheme.

The variables in the dive record are:

TIMESTAMP

The value of the internal clock (in seconds) at the beginning of the dive. See Section 2.6.

INTERVAL, THRESHOLD, ENDCOUNT, AVERAGING

The four recording parameters (see Section 2.8). These values indicate the state of the recording parameters *at the time of the dive*, which may not match the current settings.

TEMPERATURE, PRESSURE

Each data sample consists of a TEMPERATURE/PRESSURE pair, encoded as described in Section 2.7.

You can tabulate the dives in the DATA segment by using this simple algorithm:

1. Start at the beginning of the DATA segment.
2. Step forward one byte at a time until you find a dive header (0x00000000).
3. Read the header variables TIMESTAMP, INTERVAL, THRESHOLD, ENDCOUNT, AVERAGING.
4. Read pairs of TEMPERATURE/PRESSURE values until you reach the dive footer (0xFFFFFFFF).
5. Repeat Steps 2 - 5 until you reach the end of the DATA segment.

This "brute force" algorithm is not very efficient, since new dive data is usually concentrated in just the last few pages of the DATA segment. If you know the timestamp of the last dive seen during the last download, you could use a much "smarter" algorithm:

1. Start at the end of the DATA segment.
2. Step backward one byte at a time until you find a dive header (0x00000000).
3. Read the TIMESTAMP.
4. Repeat Steps 2 - 3 until you reach a dive whose TIMESTAMP has been seen before.
5. Advance forward as in the brute force algorithm to read just the new dives.

This modified algorithm is particularly useful on slower/power-conscious hosts such as PDAs.

5.0 Basic Handshaking

"Handshaking" is the term used to describe the initial communication between device and host. During this communication sequence, the device identifies itself and the host is given an opportunity to transmit an instruction for further processing.

5.1 The Handshake Packet

The device spends most of its time in a deep sleep mode to conserve power. Once a second the recorder briefly wakes up to check if it is connected to a live serial port. If so, the device transmits a **26-byte handshake packet**.

The handshake packet identifies the device type, firmware revision, recording parameters, and provides some other diagnostic data. The packet is constructed as follows:

Offset	Type	Name
0x0000	UInt16	VERSION
0x0002	UInt16	SERIAL
0x0004	UInt32	TIME
0x0008	UInt16	BOOT_COUNT
0x000A	UInt32	BOOT_TIME
0x000E	UInt16	DIVE_COUNT
0x0010	UInt16	INTERVAL
0x0012	UInt16	THRESHOLD
0x0014	UInt16	ENDCOUNT
0x0016	UInt16	AVERAGING
0x0018	UInt16	CRC

The variables in the handshake packet are described in more detail below.

VERSION

The most significant byte of VERSION identifies the major product version, and will always equal **0x03** for Sensus Ultra devices. The least significant byte identifies the firmware version, and will be **0x01 or higher**.

SERIAL

This value matches the serial number printed on the front sticker of the device. The serial number **uniquely identifies** the device and can be used to differentiate dive data acquired from multiple sources.

TIME

The current value of the internal clock in **seconds**. Refer to Section 2.6 for an explanation of the timekeeping system.

BOOT_COUNT

The number of times the internal processor has been rebooted. Typically this will be 1, indicating the factory reboot after assembly.

BOOT_TIME

The timestamp of the most recent reboot. Typically this will be 0, since the clock is reset during assembly.

DIVE_COUNT

The total number of dive records recorded by the device in its lifetime. This number does **not necessarily** represent the number of dives currently stored in memory, since older dives may have been expunged if the device has been filled.

INTERVAL, THRESHOLD, ENDCOUNT, AVERAGING

The four recording parameters, as defined in Section 2.8.

CRC

The last two bytes of the handshake packet are the 16-bit CRC of the previous 24 data bytes. You should **always verify** the integrity of the handshake packet by computing/comparing the CRC. If you discover a mismatch, simply discard the corrupted handshake packet and wait for the next one.

5.2 Flow Control Rules

To avoid overflowing the single-byte serial buffer in the device, the host must **strictly obey** certain flow control rules when transmitting data.

If/when the device is ready to receive a byte, it sends a **prompt byte (0xA5)** to the host. The host then has a **50 ms window** within which a byte may be sent to the device.

If the host ignores the prompt, the device aborts the pending operation and returns to sleep. Similarly, if the host transmits a byte outside the window of opportunity, it is ignored.

5.3 Instruction Codes

Immediately after transmitting the handshake packet, the device expects to receive a **16-bit instruction code** from the host.

The permissible instruction codes are:

Code	Instruction
0xB410	SET_INTERVAL
0xB411	SET_THRESHOLD
0xB412	SET_ENDCOUNT
0xB413	SET_AVERAGING
0xB420	READ_USER
0xB421	READ_DATA
0xB430	WRITE_USER
0xB440	SENSE

Each instruction protocol is documented in the next chapter. If no instruction code is received, or if an invalid code is transmitted, the device simply returns to sleep.

What may look like a "27th byte" of the handshake packet is really a prompt byte (0xA5) indicating the device's readiness to accept the LSB of an instruction code.

5.4 Summary of the Handshake Sequence

All communication with the device begins the same way: the host waits for the handshake packet, verifies it, and may optionally send an instruction code to initiate further dialogue with the device. The following is a brief summary of this procedure:

6. Flush the host's input/output buffers.
7. Wait until 26 bytes (the handshake packet) accumulate in the input buffer.
8. Read the handshake packet.
9. Compute the CRC of the first 24 bytes of the packet.
10. Compare the computed CRC to the CRC from the packet.
11. If the CRCs do not match, wait roughly 250 ms** and return to Step 1.
12. Wait for the prompt byte (0xA5).
13. Send the least-significant byte of the instruction code.
14. Wait for the prompt byte (0xA5).
15. Send the most-significant byte of the instruction code.

***In step 6, a 250 ms delay is suggested to guarantee that the prompt byte sent after the handshake packet is not accidentally buffered by the host and (mis)interpreted as part of the next packet.*

6.0 Instruction Protocols

This chapter describes the precise communication sequences required for each permissible instruction. If you encounter unexpected results, review Chapters 2 - 4 before contacting ReefNet for assistance.

6.1 SET_INTERVAL (0xB410)
6.2 SET_THRESHOLD (0xB411)
6.3 SET_ENDCOUNT (0xB412)
6.4 SET_AVERAGING (0xB413)

The SET series of instructions are used to modify the recording parameters INTERVAL, THRESHOLD, ENDCOUNT, and AVERAGING. The particular instruction code used selects the variable to be modified -- the instructions otherwise operate identically.

After issuing a SET instruction, the recorder expects to receive the new 16-bit value for the parameter. Once the value is transmitted (obeying the flow control rules), the device modifies the parameter and returns to sleep.

If an invalid parameter value is transmitted, the device ignores the instruction and returns to sleep. The valid ranges for each parameter are defined in Section 2.8.

Never assume that the correct value has been stored in the device! Instead, wait for the next handshake packet and check that the reported parameter values match your expectations.

6.5 READ_USER (0xB420)
6.6 READ_DATA (0xB421)

The READ instructions initiate the **page-by-page** transmission of the USER and DATA segments. After transmitting one of the READ instructions, the device responds with a **516 byte page packet**, constructed as follows:

Offset	Type	Name
0x0000	UInt16	PAGENUM
0x0002	512 bytes	PAGEDATA
0x0202	UInt16	CRC

PAGENUM indicates the position in the overall transmission sequence. The first page transmitted is page 0, the next is 1, and so forth. **PAGEDATA** is the actual data contained within the memory page. Lastly, the **CRC** value is the 16-bit CRC of PAGEDATA. The integrity of PAGEDATA should always be verified using the CRC.

After sending a page packet, the device expects the host to **accept**, **reject**, or **ignore** the packet:

ACCEPTING A PAGE PACKET

If no errors are detected in a page packet and you wish to receive the next one, respond to the prompt byte by echoing back the same byte (**0xA5**). The device will increment PAGENUM and begin sending the next page packet **within 100 ms**.

REJECTING A PAGE PACKET

If you detect an error in a page packet, you may request that the device **resend the packet** by responding to the prompt byte with the **null byte (0x00)**. The device will resend the same packet (without incrementing PAGENUM) **within 100 ms**.

You may reject a packet as many times as necessary until you get a clean transmission.

IGNORING A PAGE PACKET

If you do not respond to the prompt byte within the prescribed time window (see Section 5.2), the recorder will terminate the READ instruction and return to sleep.

READ_USER traverses the pages in the USER segment from top to bottom.

READ_DATA traverses the pages in the DATA segment from bottom to top. The first page transmitted (PAGENUM = 0) is actually the last page in the segment.

Since the pages of the DATA segment are read in *reverse-chronological order*, the host can terminate the READ operation as soon as it encounters a page that it has seen before.

In other words, rather than blindly downloading the entire DATA segment, you can (and should!) grab just the pages which contain new dive data. This might mean the difference between a 3 *minute* download and a 3 *second* download.

6.7 WRITE_USER (0xB430)

The WRITE_USER instruction is used to overwrite the USER segment of memory.

After issuing the instruction, the device expects **16384 bytes** of raw data, followed by the **16-bit CRC** of the raw data. Standard flow control rules apply: the host must wait for a prompt byte before sending each byte to the device.

Once all 16386 bytes (16384 + 2) have been transmitted, the device verifies the integrity of the data using the CRC. If the data is intact, the device copies it into the USER segment. If any transmission errors are detected, the raw data is simply discarded.

Never assume that the USER segment been written successfully! Instead, wait for the next handshake cycle and issue a READ_USER instruction. Compare the data that should have been written to the data actually received from the device.

6.8 SENSE (0xB440)

The SENSE instruction causes the device to report the current battery voltage, temperature, and pressure to the host. This feature is useful when real-time pressure or temperature data is required. For instance, the device could be connected to a computer inside a hyperbaric chamber for live display of ambient conditions.

Within 500 ms of issuing the SENSE instruction, an **8-byte sense packet** is delivered to the host. The device then returns to sleep. The sense packet is constructed as follows:

Offset	Type	Name
0x0000	UInt16	VOLTAGE
0x0002	UInt16	TEMPERATURE
0x0004	UInt16	PRESSURE
0x0006	UInt16	CRC

The encoding of VOLTAGE, TEMPERATURE and PRESSURE are defined in Section 2.7. The last two bytes of the sense packet are the 16-bit CRC of the previous 6 data bytes. You should **always verify** the integrity of the sense packet by computing/comparing the CRC.