Blueprint Subsea

# SeaTrac Developer User Guide

Distributed by ROVITEC AS

rovitec.no

Manufacturer: Blueprint Subsea

Document: UM-140-D00221-13.pdf

Source checked: 16 September 2026

Use this guide with the specified model and firmware. Manufacturer pages and revision identifiers are retained unchanged.

**Original manufacturer source**

https://www.blueprintsubsea.com/downloads/seatrac/UM-140-D00221-13.pdf

Manufacturer pages follow unchanged. Local distribution copy.

seatrac

blueprint subsea




# SeaTrac

# Developer Guide

(for Beacon firmware version 3.7)

UM-140-D00221-13

# Contents

1. Introduction ..... 7
1.1. Overview ..... 7
1.2. Document Revision and Change History ..... 9
1.3. Technical Support ..... 14
1.4. Numerical Notations ..... 15
1.5. Notices ..... 15
2. Beacon Overview ..... 17
2.1. Serial Command Processor ..... 17
2.2. Settings Manager and EEPROM Memory ..... 17
2.3. Environmental Sensor System ..... 18
2.4. Attitude & Heading Reference System ..... 18
2.5. Acoustic Transceiver & Data Modem ..... 19
2.6. USBL Processor ..... 20
2.7. Acoustic Protocol Stack ..... 20
3. System Setup ..... 21
3.1. Equipment Required ..... 21
3.2. Testing in Air ..... 21
3.3. Testing in Open Water ..... 22
3.4. Acoustic Limitations ..... 23
3.5. Diagnostic Tools ..... 23
3.5.1. PinPoint ..... 23
3.5.2. SeaTracTools ..... 24
4. Beacon Procedures ..... 26
4.1. Reading Beacon Information ..... 26
4.2. Fetching, Updating and Storing Settings ..... 27
4.3. Requesting Status Output ..... 28
4.4. Acoustically Pinging Beacons ..... 28
4.5. Calibrating the AHRS ..... 30
5. Serial Interface Protocol ..... 32
5.1. Serial Settings ..... 32
5.2. Message Format ..... 32
5.3. Synchronisation Characters ..... 33
5.4. Data Encoding ..... 33

5.5. Command ID Codes ..... 33
5.6. Checksums ..... 34
6. Message Field Type and Constant Definitions ..... 36
6.1. Primitive Types ..... 36
6.2. Arrays ..... 37
6.3. Enumerations & Constants ..... 37
6.3.1. AMSGTYPE_E (Acoustic Message Type) ..... 38
6.3.2. APAYLOAD_E (Acoustic Payload Identifier) ..... 40
6.3.3. BAUDRATE_E (Serial Port Baud Rate) ..... 41
6.3.4. BID_E (Beacon Identification Code) ..... 41
6.3.5. CAL_ACTION_E (Calibration Actions) ..... 43
6.3.6. CID_E (Command Identification Codes) ..... 44
6.3.7. CST_E (Command Status Codes) ..... 49
6.3.8. STATUSMODE_E (Status Output Mode) ..... 53
6.3.9. XCVR_TXMSGCTRL_E (Transmit Message Control) ..... 53
6.3.10. CFG_ADDRESS_E (Acoustic Config Address) ..... 54
6.3.11. BEACON_TYPE_E (Beacon Type) ..... 55
Structures ..... 56
6.3.12. ACOMSG_T (Acoustic Message) ..... 56
6.3.13. ACOFIX_T (Acoustic Position and Range Fix Summary) ..... 58
6.3.14. AHRSCAL_T (AHRS Calibration Coefficients) ..... 63
6.3.15. FIRMWARE_T (Firmware Information) ..... 66
6.3.16. HARDWARE_T (Hardware Information) ..... 67
6.3.17. IPADDR_T (IP v4 Address) ..... 68
6.3.18. MACADDR_T (MAC Address) ..... 69
6.3.19. NAV_QUERY_T (NAV Protocol Query Bit Mask) ..... 70
6.3.20. PRESSURE_CAL_T (Pressure Sensor Calibration) ..... 70
6.3.21. SETTINGS_T (Settings Record Structure) ..... 72
6.3.22. STATUS_BITS_T (Status Fields Bit-Mask) ..... 79
7. Beacon Management Message Definitions ..... 80
7.1. System Messages ..... 80
7.1.1. CID_SYS_ALIVE ..... 80
7.1.2. CID_SYS_INFO ..... 81
7.1.3. CID_SYS_REBOOT ..... 83
7.1.4. CID_SYS_ENGINEERING ..... 84
7.2. Firmware Programming Messages ..... 85
7.2.1. CID_PROG_INIT ..... 86
7.2.2. CID_PROG_BLOCK ..... 88
7.2.3. CID_PROG_UPDATE ..... 90
7.3. Status Messages ..... 91

7.3.1. CID_STATUS ..... 91
7.3.2. CID_STATUS_CFG_GET ..... 97
7.3.3. CID_STATUS_CFG_SET ..... 98
7.4. Settings Messages ..... 99
7.4.1. CID_SETTINGS_GET ..... 99
7.4.2. CID_SETTINGS_SET ..... 100
7.4.3. CID_SETTINGS_LOAD ..... 101
7.4.4. CID_SETTINGS_SAVE ..... 102
7.4.5. CID_SETTINGS_RESET ..... 103
7.5. Calibration Messages ..... 104
7.5.1. CID_CAL_ACTION ..... 104
7.5.2. CID_AHRS_CAL_GET ..... 106
7.5.3. CID_AHRS_CAL_SET ..... 107
7.6. Acoustic Transceiver Messages ..... 108
7.6.1. CID_XCVR_ANALYSE ..... 108
7.6.2. CID_XCVR_TX_MSG ..... 110
7.6.3. CID_XCVR_RX_ERR ..... 111
7.6.4. CID_XCVR_RX_MSG ..... 113
7.6.5. CID_XCVR_RX_REQ ..... 114
7.6.6. CID_XCVR_RX_RESP ..... 115
7.6.7. CID_XCVR_RX_UNHANDLED ..... 116
7.6.8. CID_XCVR_USBL ..... 117
7.6.9. CID_XCVR_FIX ..... 119
7.6.10. CID_XCVR_STATUS ..... 120
7.6.11. CID_XCVR_TX_MSGCTRL_SET ..... 121
7.6.12. CID_XCVR_USBL_TIME ..... 123
7.6.14. CID_XCVR_BASELINES ..... 124
8. Acoustic Protocol Stack Message Definitions ..... 125
8.1. PING Protocol Messages ..... 125
8.1.1. CID_PING_SEND ..... 126
8.1.2. CID_PING_REQ ..... 128
8.1.3. CID_PING_RESP ..... 128
8.1.4. CID_PING_ERROR ..... 129
8.2. ECHO Protocol Messages ..... 130
8.2.1. CID_ECHO_SEND ..... 130
8.2.2. CID_ECHO_REQ ..... 132
8.2.3. CID_ECHO_RESP ..... 133
8.2.4. CID_ECHO_ERROR ..... 134
8.3. DAT Protocol Messages ..... 135
8.3.1. CID_DAT_SEND ..... 136
8.3.2. CID_DAT_RECEIVE ..... 138

8.3.3. CID_DAT_ERROR ..... 140
8.3.4. CID_DAT_QUEUE_SET ..... 141
8.3.5. CID_DAT_QUEUE_CLR ..... 144
8.3.6. CID_DAT_QUEUE_STATUS ..... 145
8.3.7. CID_DAT_QUEUE_GET ..... 146
8.4. NAV Protocol Messages ..... 147
8.4.1. CID_NAV_QUERY_SEND ..... 148
8.4.2. CID_NAV_QUERY_REQ ..... 150
8.4.3. CID_NAV_QUERY_RESP ..... 151
8.4.4. CID_NAV_ERROR ..... 154
8.4.5. CID_NAV_QUEUE_SET ..... 155
8.4.6. CID_NAV_QUEUE_CLR ..... 157
8.4.7. CID_NAV_QUEUE_STATUS ..... 158
8.4.8. CID_NAV_STATUS_SEND ..... 159
8.4.9. CID_NAV_STATUS_RECEIVE ..... 161
8.4.10. CID_NAV_QUEUE_GET ..... 162
8.5. CFG Protocol Messages ..... 163
8.5.1. CID_CFG_BEACON_GET ..... 164
8.5.2. CID_CFG_BEACON_SET ..... 165
8.5.3. CID_CFG_BEACON_RESP ..... 167
8.6. CFG Protocol Version 2 Messages ..... 168
8.6.1. CID_CFG_REQ_SEND ..... 169
8.6.2. CID_CFG_REQ_RECEIVE ..... 171
8.6.3. CID_CFG_RESP_SEND ..... 172
8.6.4. CID_CFG_RESP_RECEIVE ..... 173
8.6.5. CID_CFG_LOCAL_LOCK ..... 174
8.6.6. CID_CFG_LOCAL_UNLOCK ..... 175
8.7. TRACK Protocol Messages ..... 176
8.7.1. CID_TRACK_SEND ..... 177
8.7.2. CID_TRACK_REQ ..... 178
8.7.3. CID_TRACK_RESP ..... 178
8.7.4. CID_TRACK_ERROR ..... 179
8.8. PINGER Protocol Messages ..... 180
8.8.1. CID_PINGER_SEND ..... 180
8.9.1. CID_PINGER_SET_ENABLED ..... 181
8.11. INTERRUPT Protocol Message ..... 182
8.11.1. CID_INTERRUPT_SEND ..... 182
8.11.2. CID_INTERRUPT_RECEIVE ..... 183
8.11.3. CID_INTERRUPT_RESET_IDLE ..... 183
8.11.4. CID_INTERRUPT_UNLOCK ..... 183
9. Beacon Definitions and Frames of Reference ..... 184

9.1. Attitudes (Yaw, Pitch and Roll) ..... 184
9.2. USBL Spherical Angles (Azimuth and Elevation) ..... 185
9.3. USBL Local Relative Position Coordinates ..... 186
9.4. USBL World Relative Position Coordinates ..... 186

# 1. Introduction

## 1.1. Overview

The SeaTrac X100 series of Micro-USBL tracking and data modems are suite of complimentary products built around a robust broadband spread spectrum signalling scheme. These multi-purpose acoustic transponder beacons are capable of simultaneously tracking asset positions and undertaking bi-directional data exchange, making them ideal for use in a wide range of applications including…

- Remote monitoring and control of sensors and equipment,
- Re-location and retrieval of sub-sea assets,
- ROV positioning and navigation tasks,
- AUV navigation, telemetry, mission adjustments and real-time position monitoring,
- Diver buddy and surface-vessel/dive-bell tracking and re-location,
- Remote and local depth, water temperature, attitude and heading reference (AHRS) information.

This document is intended to provide developers with the necessary information to allow interfacing of SeaTrac Beacons with their systems using the serial communications port hardware and command protocol and intended to be read in addition to the hardware and operational notes discussed in the specific Beacon's user manual.

This Interface Control Document (ICD) provides a complete reference for the low-level serial messages accepted and generated by the SeaTrac beacons, including descriptions of the data types, constants and structures that are required to encode and decode messages.

It is assumed that the reader has a reasonable level of programming expertise and is familiar with the basic concepts of the serial port hardware within their chosen operating system, as well as manipulation of numerical data types and basic software design principles.

Throughout this document the following symbols are used to indicate special precautions or procedures you should note…



**WARNING!**
This symbol indicates a warning you should follow of to avoid bodily injury and damage to your equipment.



**CAUTION**
This symbol denotes precautions and procedures you should follow to ensure correct operation to your equipment, and in some situations (where noted) possible damage.



**NOTE**
This symbol denotes special instructions or tips that should help you get the best performance from your beacons.

## 1.2. Document Revision and Change History

This section details the changes that have been made to this document as a result of new beacon firmware releases.



Developers upgrading beacon firmware should use the following summary list to check for compatibility in command message encoding and decoding algorithms.



For details on upgrading Beacon firmware, please refer to the “Upgrading Beacon Firmware” document (ref UM-140-P00916) available as part of the Beacon Firmware download package available from the Technical Support website (see section 1.3).

### Revision 13 (Firmware v3.7 upwards)

Document revised for firmware release 3.7 that:

- CID_NAV_QUEUE_GET and CID_DAT_QUEUE_GET messages added to retrieve the queued data for a remote beacon.

### Revision 12 (Firmware v3.6 upwards)

Document revised for firmware release 3.6 that:

- Adds “SUPPORTS_CFG_V2” parameter to the CID_CFG_BEACON_RESP message, indicating that the beacon supports the CFG V2 protocol.

### Revision 11 (Firmware v3.5 upwards)

Document revised for firmware release 3.5 that:

- Adds INTERRUPT protocol commands to control the generation of an interrupt tone that can be used halt a system.
- NAV_QUERY_T flags modified to include QRY_DATA_PLUS bit. When this bit is set, any queued data for the NAV protocol will be retrieved, alongside data for other flags that are specified.
- Adds version 2 of the Acoustic Configuration protocol. Which allows for more flexible control over the original CFG protocol. It also allows for configuration of devices connected to the beacon’s serial port.
- The CFG protocols can now be locked/unlocked using the CFG V2 protocol.
- When the beacon ID is set acoustically, using either version of the CFG protocol, the ID will not persist after a power cycle, leaving the beacon with an invalid ID.

### Revision 10 (Firmware v3.3 upwards)

Document revised for firmware release 3.3 that:

- Increases the maximum beacon ID to 100

### Revision 9 (Firmware v3.2 upwards)

Document revised for firmware release 3.2 that:

- Adds a new TRACK protocol.

### Revision 8 (Firmware v2.6 upwards)

Document revised for firmware release 2.6 that:

- (experimental) Adds CID_PINGER_SEND command to trigger a 37kHz CW, 10ms pulse.

### Revision 7 (Firmware v2.4 upwards)

Document revised for firmware release 2.4 that:

- New enumeration XCVR_TXMSGCTRL_E defined.
- Adds “XCVR_TX_MSGCTRL” Transmit Message Control flag bits to the XCVR_FLAGS field of the SETTINGS_T structure, that uses the XCVR_TXMSGCTRL_E enumeration values.
- Command CID_XCVR_TX_MSGCTRL_SET has been added to support direct setting and querying the state of the XCVR_TX_MSGCTRL state.
- Added PRESSURE_CAL_T structure.
- Modified CID_SYS_INFO to support extended messages with Pressure Sensor information.
- BEACON_DEST_ID and BEACON_SRC_ID fields appended to the end of CID_XCVR_USBL messages to allow for diagnostic of which beacon sent and received the USBL signals.

### Revision 6 (Firmware v2.2 upwards)

Document revised for firmware release 2.2 that:

- Fixes a bug in the acoustic receiver engine where ‘enhanced position’ fixes did not take the ‘depthLocal’ measurement into account and could cause an inaccuracy in horizontal ranges.
- Changes the acoustic properties of the acoustic signal from which USBL position measurements are made (from an 8kHz bandwidth to a wider 16kHz bandwidth giving better accuracy position measurements).
- Introduces a different acoustic symbol for message response acknowledgements from those being sent, this improves reliability in enclosed water situations (tanks, swimming pools, shallow channels) where the sent message could be received before the response if reflected back by the environment under certain circumstances.

None of the above firmware charges modify the serial command set from the previous v1.11 firmware release.

### Revision 5 (Firmware v1.11 upwards)

Document revised for firmware release 1.11, with changes to some of the NAV protocol command-set.

- Command CID_DATA_QUEUE_SET now supports a queue for each individual beacon (opposed to one queued packet for one beacon only at a time).
- Command CID_DATA_QUEUE_SET now supports clearing down of a packet for a beacon, by setting the length to 0.
- Command CID_DATA_QUEUE_SET will load the packet into all beacons queues if DEST_ID of BEACON_ALL (0) is used. Likewise using this and a length of 0 will clear all queues.
- The response from CID_DATA_QUEUE_SET now includes additional echoing of DEST_ID and the number of bytes currently queued against the specified beacon. These bytes are appended at the end of the message allowing backward compatibility.

- Using CID_DATA_QUEUE_SET with no PACKET_LEN parameter specified will take no action, but just return the amount of data currently on the queue for the specified beacon. This won't work with DEST_ID of BEACON_ALL (0), in which case a length of 0 is returned.
- Command CID_DATA_QUEUE_CLR now supports an optional parameter DEST_ID that specifies which queue to clear. Omitting the parameter or using BEACON_ALL (0) will clear all queues, this allows backwards compatibility.
- Response from CID_DATA_QUEUE_CLR now includes an echo of the DEST_ID specified in the command.
- Command CID_DATA_QUEUE_STATUS modified to now return an array of PACKET_LENGTH values for each of the 16 beacons addresses (0-15, array index 0 always reads as 0 for the BEACON_ALL position).
- Note added to the DAT protocol making it clear that since firmware 1.6 the protocol supports 'packet sniffing' and CID_DATA_RECEIVE messages will be generated for every acoustic message the beacon receives. The application developer should use the SRC_ID and DEST_ID fields to determine if and how to use the data payload in their system.

- NAV_QUERY_T flags modified to include QRY_DATA bit. When this bit is set, any queued data for the NAV protocol will be retrieved in preference to other information requested by the remaining query flags.
- CID_NAV_QUERY_SEND command modified to include additional fields for PACKET_LEN and PACKET_DATA. Both these new parameters are optional (for backward compatibility) and if omitted from the serial command will be treated as zero length. (NB: The NAV protocol PACKET_DATA has reduced length compared to PACKET_DATA fields in other protocols, like ECHO and DAT).
- CID_NAV_QUERY_REQ message modified to include PACKET_LEN and DATA fields that are used to output received data sent through the CID_NAV_QUERY_SEND command.
- CID_NAV_QUERY_RESP message modified to include optional MSG_LEN and MSG_DATA fields that will only be present when the QRY_MSG bit is set.
- Commands CID_NAV_REF_POS_SEND, CID_NAV_REF_POS_UPDATE, CID_NAV_BEACON_POS_SEND and CID_NAV_BEACON_POS_UPDATE have been deprecated – see CID_NAV_STATUS_SEND and CID_NAV_STATUS_RECEIVE for replacements.
- CID_NAV_STATUS_SEND command added (using one-way broadcast) to allow information about beacons to be broadcast to everyone (such as position etc.)
- CID_NAV_STATUS_RECEIVE message added that is output when a NAV_STATUS one-way message is received.
- Commands CID_NAV_QUEUE_SET, CID_NAV_QUEUE_CLR and CID_NAV_QUEUE_STATUS added to allow message data to be remotely queued up ready for response when the next CID_NAV_QUERY arrives with the QRY_DATA flag set.
- NAV protocol now supports acoustic packed 'snooping/sniffing' similar to the DAT protocol. When any NAV packet is received it will be output using the appropriate CID_NAV_QUERY_REQ or CID_NAV_QUERY_RESP message, but the LOCAL_FLAG field will only be true for packets intended for that beacon, and false for all 'snooped' packets.

- DEX protocol has been deprecated and is no longer available (frees RAM for the additional message queues in the DAT and NAV protocols).

### Revision 4 (Firmware v1.2 upwards)

Clarified the instructions for computing the CRC16 message checksums, as binary message contents should be used after ASCII-Hex character pairs have been converted into their byte-wise values.

### Revision 3 (Firmware v1.2 upwards)

Document revised for firmware release v1.2.

- For standard USBL response fix calculations, ACOFIX_T structure POSITION_DEPTH values are now given relative to the surface and not the beacon (i.e. local depth added to USBL resolved depth)
- For enhanced USBL response fix calculations, ACOFIX_T structure POSITION_NORTHING and POSITION_EASTING values are computed based on the remote depth, range and azimuth angle (not from standard USBL coordinates as used previously).
- SETTINGS_T structure modified to contain new Position Filter enable flag and control fields – XCVR_POSFLT_ENABLE, XCVR_POSFLT_VEL, XCVR_POSFLT_ANG and XCVR_POSFLT_TMO. Fields are appended on the end of the structure for backward compatibility.
- ACOFIX_T structure flags field modified to include POSITION_FLT_ERROR bit. This indicates when the position filter (if enabled) has determined that a fix position may be invalid based on the beacons last position, defined movement limits and time between fixes.
- Change made to the CID_NAV_BEACON_POS_UPDATE and CID_NAV_REF_POS_UPDATE message outputs, to include an ACOFIX_T field.
- Documentation now completed for the NAV protocol.
- USBL detection algorithm to reduce false detects in multi-path dominant environments.

### Revision 2 (Firmware v1.1)

Document revised for firmware release v1.1.

- Enumeration and structure definition changes include…
  - New ACOFIX_T structure definition added used by several commands.
  - Six new CST_XCVR_STATE… codes have been added for use with the CID_XCVR_STATUS command.
  - DAT_MODE_E enumeration removed as not required.
- Command message changes include…
  - The 32-bit FLAGS field in the HARDWARE_T structure has been split into two 16-bit fields, representing system (factory set) flags, and user flags.
  - CID_XCVR_RX_RESP_ERROR removed and functionality merged with CID_XCVR_RX_ERR message.

- CID_XCVR_RX_ERR message contains the new ACOFIX_T structure describing the received acoustic signal.
- CID_XCVR_RX_MSG, CID_XCVR_RX_REQ and CID_XCVR_RX_RESP include ACOFIX_T structure describing the received acoustic signal.
- CID_XCVR_FIX command now uses the new ACOFIX_T structure.
- New CID_XCVR_STATUS command added to allow the current status of the acoustic transceiver to be queried – responses include Idle, Transmitting, Receiving Header, Receiving Data etc.
- CID_PING_SEND command now has a MSG_TYPE field to specify how messages are sent – either as Request Standard (MSG_REQ), Request USBL (MSG_REQU) or Request Enhanced USBL (MSG_REQX) types. This allows finer control over acoustic transmission types and timings.
- CID_PING_REQ and CID_PING_RESP messages now contain the new ACOFIX_T structure – this means the XCVR_FIX_MSGS flag is no longer required to be set in the acoustic transceiver settings.
- CID_ECHO_SEND command now has a MSG_TYPE field to specify how messages are sent – either as Request Standard (MSG_REQ), Request USBL (MSG_REQU) or Request Enhanced USBL (MSG_REQX) types. This allows finer control over acoustic transmission types and timings.
- CID_ECHO_REQ and CID_ECHO_RESP messages now contain the new ACOFIX_T structure – this means the XCVR_FIX_MSGS flag is no longer required to be set in the acoustic transceiver settings.
- CID_DAT_SEND command now has a MSG_TYPE field (replacing FLAGS field and obsolete DAT_MODE_E enumeration) to specify how messages are sent.
- CID_DAT_SEND command DEST_ID parameter now allows a value of BEACON_ALL (0) to send data to all beacons. However, the MSG_TYPE type must be either MSG_OWAY or MSG_OWAYU.
- New commands CID_DAT_QUEUE_SET, CID_DAT_QUEUE_CLR and CID_DAT_QUEUE_STATUS added to allow data to be queued at a remote beacon, and transmitted back in a response.
- Command format for CID_DAT_RECEIVE updated to use the new ACOFIX_T structure (support beacon fix information, acknowledge flag and is generated upon receipt of an acknowledge message).
- CID_DAT_ACK message removed, as CID_DAT_RECEIVE now includes this functionality.

- Firmware bug fixed in response turnaround timing where a timing error caused a range error of up to 1m to be reported.
- Transmitter waveform modified for improved performance.
- Widened USBL detection window to allow greater multipath tolerance as signal is processed by the data decoder.

### Revision 1 (Firmware v1.0)

Initial documentation release for beacon firmware v1.0.

## 1.3. Technical Support

For the latest software and firmware updates, as well as production information, manuals and datasheets, visit the support pages at www.blueprintsubsea.com/seatrac.

For further Technical Support on either software of hardware related issues please use the contact details provided on the website above or in the Beacon's user manual, and where possible please have the products part number, serial number, firmware and software revision details available.

We welcome any feedback you may have about SeaTrac products, from bug reports to ideas for new features or hardware to support – please use the contact details on the website (or shown below) to get in touch.

## 1.4. Numerical Notations

Throughout this document references are made to numerical values expressed in decimal, hexadecimal or binary notation. To identify to base of numbers, the following notation is used…

- **Decimal** — Values in decimal are expressed in plain numerical digits without any form or prefix or postfix notation.
- **Hexadecimal** — Values expressed in hexadecimal notation are always prefixed with an "**0x**" notation; with uppercase characters for "A" to "F" (although the SeaTrac Beacons serial interface will accept both uppercase and lower case unless otherwise stated).
- **Binary** — Values expressed in a binary notation are always prefixed with an "**0b**" notation, followed by a string of "1" and "0" characters.

## 1.5. Notices

### Specifications and Content

The contents of this document are provided on an "as is" basis and although we try to ensure the information presented here is correct at the time of going to press, this document may contain some errors. Blueprint Design Engineering Ltd cannot be held responsible for any inaccuracies or omissions. If you find an error or feel we have missed important or useful information, please contact us. The latest version of this document is always available to download from the website.

Specifications and information contained in this document are subject to change at any time without notice, and does not represent a commitment on the part of Blueprint Design Engineering Ltd.

Where possible, we will aim to re-release this document in conjunction with significant firmware releases and updates, and will document the changes made to the document's technical contents (commands, data structures, message formats etc) in the "

Document Revision and Change History".

### Disclaimer

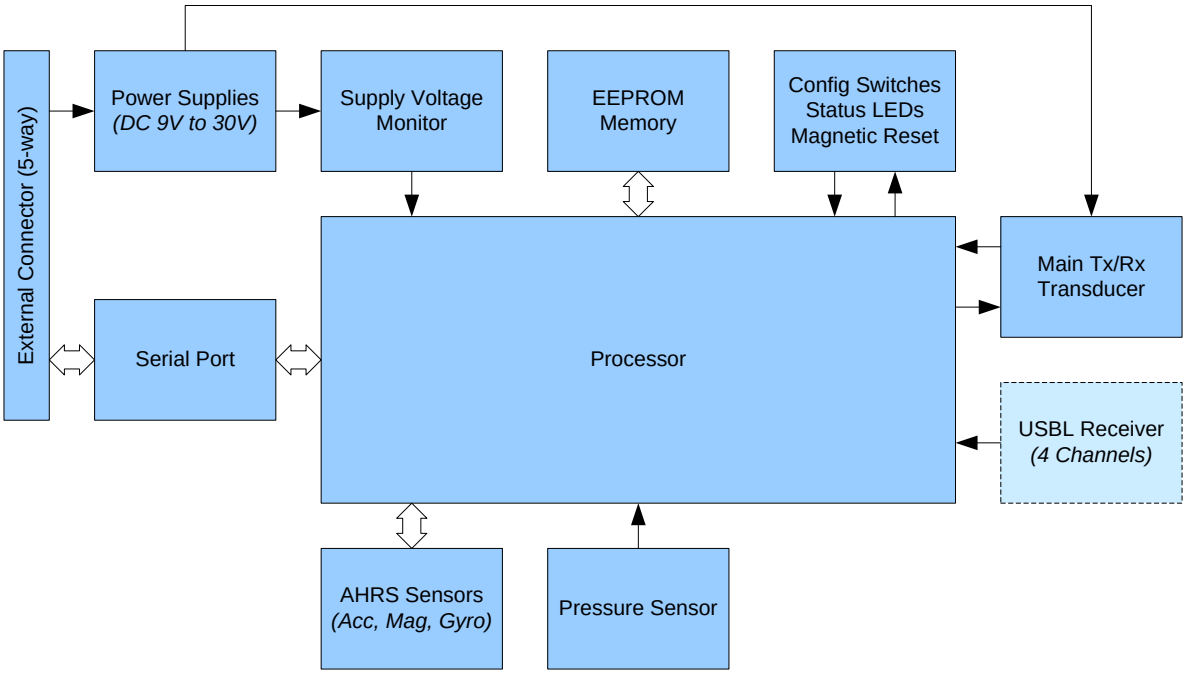
Neither Blueprint Design Engineering Ltd., or their affiliates shall be liable to the purchaser of this product, or third parties, whether in tort, contract or otherwise for losses, costs, damages or expenses incurred by the purchaser or third parties as a result of accident, misuse, abuse, modification of this product or a failure to strictly comply with the operating and maintenance instructions.

### Trademarks

The Windows™ operating system is a trademark of the Microsoft Corporation. Other product and brand names used within this document are for identification purposes only. Blueprint Design Engineering Limited disclaims any and all rights in those marks.

# 2. Beacon Overview

The diagram below shows the hardware blocks in each beacon…




The X150 and X110 hardware are very similar, with the exception that X110 beacons do not have the USBL receiver circuitry or transducers fitted.

## 2.1. Serial Command Processor

The serial command processor sub-system consists of a series of routines running on the processor to decode incoming serial data into messages, ensure that correct formatting is observed and the message content is valid by means of a checksum.

The serial UART hardware features a 64-byte hardware receive buffer, from which data is analysed and decoded started on reception of the correct “synchronisation character”. Decoding and buffering of the command continues until the end of the message is reached at which time it is executed by the relevant command handler function.



Further details of the serial command protocol are discussed in section 5 from page 7 onwards.

## 2.2. Settings Manager and EEPROM Memory

Operational settings for the Beacon are stored in permanent non-volatile EEPROM memory and are loaded into a working RAM copy on power up.

Serial commands are provided to allow the user to mage changes to the working RAM settings or read them back, and to save of load the RAM settings back into EEPROM as required.

However, like other memories of its type, the EEPROM memory has a lifetime endurance of between 10,000 and 50,000 cycles, after which its data retention capabilities may start to degrade. For this reason, it is recommended that the settings are only saved by to EEPROM following changes by the user, but not on an automated periodic basis.

Some settings values can be automatically computed/changed as the beacon operates, such as pressure-sensor offsets, velocity-of-sound and magnetic calibrations. When the beacon is powered off, these changes will be lost unless specifically committed to EEPROM with a save command. In practice though, on dynamic platforms it is often found that storing a default calibration for the magnetometer is sufficient for start-up accuracy and the dynamic calibration routine will further adjust the calibration as required by the magnetic environment. Values such as pressure offset and VOS are computed within the first couple of seconds of operation.

Some settings, such as those controlling the communication hardware, are only applied on power-up or a software reboot command. Details of these are discussed in section 6.3.21 on page 72.

Settings can be reset to factory values by either issuing a Reset serial command, or using the magnetically triggered reset-to-default sequence – for further details refer to the Beacons user manual.

## 2.3. Environmental Sensor System

Each beacon is fitted with an environmental pressure and temperature sensor that allow the depth of each beacon to be calculated and monitored.
When used as part of a tracking system, the remote beacon's depth information can be transmitted and used as part of the position solution, improving vertical accuracy.
The pressure and temperature information can also be used to automatically update the local velocity-of-sound value at each beacon, ensuring ranging calculations have the least possible error.

Additionally, each beacon has a supply voltage monitoring circuit that can be used to provide low-voltage alarms for battery powered systems. The data modem feature can be used to optionally broadcast this information to other beacons on demand.

## 2.4. Attitude & Heading Reference System

Each beacon is fitted with a 9 Degrees-of-Freedom (9-DOF) Attitude and Heading Reference System, processing data from the onboard MEMS gyroscope, accelerometer and magnetometer to compute pitch, roll and yaw information that is made available to external applications via the communications port.

The AHRS sensor contains a magnetometer that is sensitive to distortion from steel structures around it (i.e. vessel hull, harbour pilings etc), and may require calibration before use.
For further details on calibration procedures, refer to the PinPoint software manual or Beacon user manual.
Section 7.5 (from page 104) in this document deals with calibration configuration messages.

## 2.5. Acoustic Transceiver & Data Modem

The Acoustic Transceiver module provides control over the beacons transmitters and receivers allowing packets of data to be sent and received by adding and validating appropriate header and checksum information.

Each beacon is configured by the user with a unique identification-code that allows up to 100 beacons to exchange acoustic data messages or broadcast to all other beacons in the network.
Messages are exchanged by a request/response process and when complete the sending beacon is able to obtain timing information and a range measurement for the remotely interrogated beacon.

When data is sent, the transceiver will use one of the message types are available…

| | |
|---|---|
| • One-Way | Data is sent One-Way, and does not require a response. (so may be broadcast to all other beacons if required). |
| • One-Way USBL | As above, data is sent one way, but additional USBL information is sent allowing the the receiver X150 beacon to compute an incoming bearing, although range cannot be provided. |
| • Request<br>• Request USBL<br>• Request USBLX (Enhanced) | Data is sent from the sender to a remote beacon, and requires that a response is returned within a set time limit (or a timeout will occur).<br>From the overall trip time ranging information can be computed, and combined with the USBL or Enhanced information, a relative position of the remote beacon can be computed. |
| • Response<br>• Response USBL<br>• Response USBLX (Enhanced) | When a message of the above type is received, response messages are sent back to complete the transaction.<br>From this state machine in protocol handlers can determine suitable actions to take. |

## 2.6. USBL Processor

The X150 beacons feature an Ultra-Short Baseline (USBL) receiver array capable of calculating the azimuth and elevation of incoming acoustic signal, and combining this information with range and the AHRS system can compute the position of the remote beacon in relative real-world coordinates (Northing, Easting and Depth).

Tracking and navigation systems can be built using one X150 is mounted from the supervisor vessel, with an application controlling the sequential 'pinging' of remote beacons. All relative positions are computed internally and output the results via serial messages, so no additional PC hardware is required.
In this mode up to 14 beacons may be tracked with the position of each being optionally broadcast to others in the network, using the data modem feature.
Alternately, developers may use several X150 beacons fitted to divers or other subsea assets and establish their own control algorithms to allow create a multiple access network where each asset can "ping" and obtain positions for every other beacon.

## 2.7. Acoustic Protocol Stack

The Acoustic Protocol Stack is connected to the previously described Acoustic Transceiver module and provides higher-level additional beacon functionality through the use of Acoustic Protocols (suites of similar commands).
The supported beacon protocols are…

- PING — Simple Ping protocol providing the basic ability to query if a beacon is powered-up and on its response obtain a range and position for it.
- ECHO — Testing and diagnostic protocol allowing packets of data to be sent to a remove beacon and returned back to the sender.
- NAV — Navigation protocol, building on the ranging and positioning capabilities of the beacon to add support for querying remote sensor information, obtaining enhanced position fixes, and broadcasting beacons positions to other users of the network.
- DAT — Datagram protocol providing commands allowing simple packets of data to be sent to a remote beacon, and an optional acknowledgment return made.
  From this developers can implement their own data exchange protocols.
  Position and ranging information are available for each acknowledgment received.



For further details on Acoustic Protocols and their command sets, please refer to section 8 from page 125.

# 3. System Setup

When using SeaTrac Beacons for the first time, ensure that you have read the Beacon's User Manual first and observed all the operating requirements and precautions.
Specially, the user manual covers the connector pin-outs used for making connections to the beacon and the requirements of the power supply.

## 3.1. Equipment Required

For a test and development setup it is recommended to have the following equipment…

- A PC with at least one RS-232 serial port, although dual serial ports are better for testing both ends of an acoustic link with two beacons (a variety of RS232-to-USB serial converters have been tested at 115200 baud and proved to also work well).
- If the PC has Microsoft Windows installed, having a copy of SeaTrac Tools installed is also recommended to help validate Beacon hardware is functioning correctly and application development progresses.
- A terminal application running on the PC to validate basic operation and view output from the beacons serial port.
- For each beacon in the setup, a separate bench-top power supply capable of delivering at least 1 amp at 12V with a smoothed DC output.
- Suitable leads to connect the beacon (or beacons) to the power supplies and RS232 serial ports.
- A bucket of water to place beacons while communicating acoustically (observe safe handling procedures when using electrical items in the proximity of mains power supplies!).

## 3.2. Testing in Air

As mentioned above, where possible it is recommended to place beacons in a bucket of water (or larger test tank is possible) while they are communicating acoustically.

The presence of the liquid around the transducer helps mechanically dampen it to its designed operating levels and will reduce peak-current observed to be drawn from power supplies during transmission.

Alternately, two separate small containers of water may be used to hold each beacon, and operation should still be observed as the sound leaves one container, travels through air, and enters the second.



Where possible, it is recommended to avoid prolonged operation of the beacons acoustic transmitter out of water as the mechanical stress on the transducer element is increased (due to reduced mechanical damping).

Taking the above note into account, development and testing of the beacons in air is still possible so long as the developer is aware of the risks.

Beacons can be placed on a bench about 30cm to 50cm apart and should communicate acoustically with each other, although sometimes it's more reliable to use clamp-stands to hold them away from the surface of the desk.

Developers should also be aware that in close proximity in air, there can be electro-magnetic cross-talk and triggering between beacons (and their cabling) rather than successful receipt of acoustic messages.



If the velocity-of-sound value for the transmitting beacon is manually specified to be around 340ms$^{-1}$, then ranges accurate to around ±20mm should be obtained.
However, accurate and repeatable USBL positioning is not possible in air due to the geometry of the USBL receiver array and the wavelength of sound in air.

## 3.3. Testing in Open Water

If testing/developing a system in open water, including small volumes (such as test-tanks, swimming pools, quarries etc) and larger volumes (lakes, sea), please bear in mind the following points when setting up the equipment:

- **Place free hanging beacons (i.e. the X150) at least 1m under-water, 1m above the sea-bed, and 1m horizontally away from any other objects (side of test tank, harbour/quay side, jetty, piling etc).**

  This is because SeaTrac beacons use acoustic "Chirp" signalling to help reject multi-path signals generated by sound reflections (i.e. front the water surface, seabed, sides of a tank, other objects etc.), but it requires an initial echo separation of at least 1m.

- **Ensure the volume of water is large enough.**

  As SeaTrac beacons are designed to operate over ranges of up to 1km with long data messages (0.5s up to 3s), their transmissions can overwhelm (saturate) the receivers of other beacons if they are positioned closely together or in a small volume of water (i.e. a test tank) that doesn't allow the acoustic energy to dissipate away from the transmitter.

  Typically, the minimum recommended size of test tank would have horizontal dimensions of at least 5 to 10m square, and be at least 2m deep.

- **Ensure mounted beacons (i.e. on a Diver, ROV, AUV etc) have a direct "line-of-sight" to other SeaTrac beacons.**

  The ranging and positioning functions of SeaTrac beacons require the shortest (most direct) acoustic path to be used. Any obstruction of this path may cause other SeaTrac beacons to receive a reflected signal (i.e. bounced off the seabed or other objects), and this will cause inaccurate ranges and positions to be reported.

- **Check the Salinity settings have been adjusted for the type of water (i.e. 0ppt for fresh-water, or 35ppt for generic seawater).**

  The Salinity setting is used to compute the velocity-of-sounds in water, and this in turn is used in range calculations. An incorrect salinity value will lead to errors in range readings.

- **Check there are no other acoustic transmitters in the water (operating below 100kHz).**

If testing in harbours, rivers, close to other vessels or from a vessel, there is a good chance that other echosounders/depth-sounders may be in operation. Operation close to equipment that is transmitting at frequencies below 100kHz may affect the performance of the SeaTrac system, and transmissions in the 20kHz-36kHz band will conflict with SeaTrac's own transmissions.

## 3.4. Acoustic Limitations

The operational ranges and performance that can be achieved between SeaTrac acoustic beacons depends heavily on environmental factors (including water temperature, dissolved oxygen content, marine plant life and man-made acoustic noise from passing vessels and submerged equipment), geographical conditions (such as depth of channel, acoustic reverberation nearby structures and obstacles that may block the acoustic transmission) and the mounting method, orientation and position of the beacon of each operating platform.

Acoustic performance values stated in the documentation describe achievable results obtained in favourable conditions, including:

- Open water with no impurities (solids or gasses in suspension)
- Consistent temperature throughout the transmission channel (no thermoclines causing refraction effects that reduce operating range)
- No other <100kHz acoustic transmission sources in the water

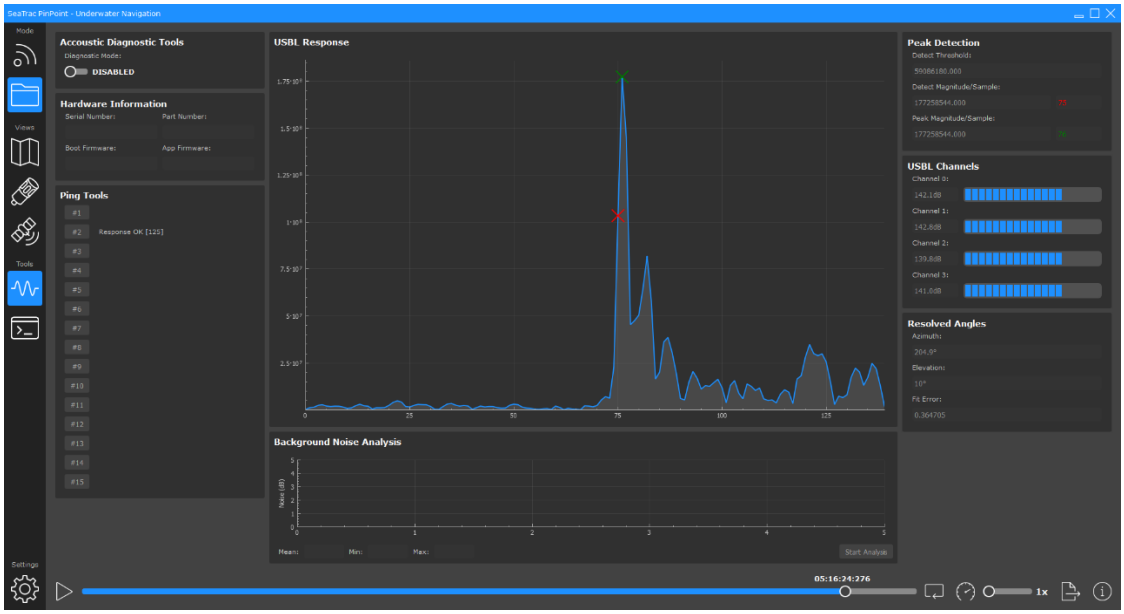
## 3.5. Diagnostic Tools

### 3.5.1. PinPoint

PinPoint is the primary software application supplied with SeaTrac beacons (and available for download from the Blueprint website) that provides an out-of-the-box tracking solution, where remote beacons (X110/X010 models) are sequentially interrogated by a single X150 USBL head attached to a Windows PC running the software. Beacon positions are shown on a top-down 2D chart overlaid onto a variety of map imagery.
PinPoint also provides tools to calibrate the AHRS heading sensors in the SeaTrac X150 beacons, and diagnostic displays to show the reception of acoustic signals in the X150 USBL head.

#### USBL Response

The diagnostic display below shows reception of a typical USBL signal, with a single good clean peak in the middle of the display, and minimal "multi-path" echoes occurring after it…




### Pinging Beacons

Additional tools are provided on the left of the display to allow interrogation of individual Beacons by acoustically "Pinging" them (a basic ranging function with no data exchange).

### Background Noise Analysis

The bottom display allows a "Background Noise Analysis" to be performed, where the receiver just listens to other acoustic signals in the water. This is useful to determine is other echosounders are operating in the area, or engine-noise/vibrations from the boat operations are being undertaken from are affecting performance.

Click the "Start" button and check that background noise levels are typically below 100dB.

The higher the background noise level is, the less ability the system will heave to receive quieter signal from beacons further away.
Typically signals from beacons close to the transmitter with can have a maximum reported received signal strength (RSSI) is 137dB, while signals from beacons at about 1km will be down at 100-105dB.

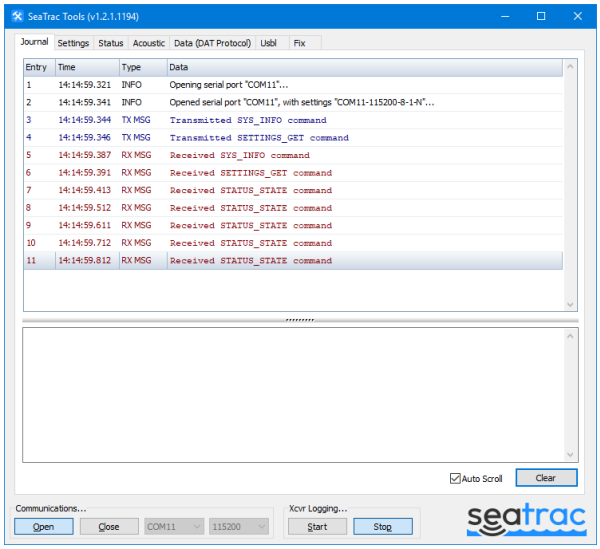
## 3.5.2. SeaTracTools

In addition to PinPoint, the "SeaTracTools" application is also installed as part of the PinPoint software installation.

SeaTrac tools is provided as an engineering diagnostic application only and not intended for day-to-day operational use, as it provides access to most of the settings and functions described in this document.
**However, care should be taken when using this application as settings can be adjusted that may degrade the performance of the Beacon's or cause inaccurate range and position readings to be reported.**

In SeaTracTools, the graphical interface provides user controls and displays to send messages to and from the beacon hardware, and the communications "Journal" function keeps track of the individual serial commands sent and received.

Clicking on each entry in the Journal will expand the message, showing the encoding/decoding order of each message data field and its corresponding value(s) – this information can be used in conjunction with the message reference descriptions provided in sections 7 and 8 to analyse and debug the operation of the Beacon.



# 4. Beacon Procedures

## 4.1. Reading Beacon Information

For new users, it is recommended that the first activity attempted is to read and validate the beacon information command.

- First, connect a beacon to the computers serial port and run a serial terminal application to establish a connection (several free “serial terminal” applications are available to download from the internet).
- If required set the terminal application to echo characters typed locally and append line-feeds to carriage-return characters (so a <CR><LF> character pair is sent at the end of messages).
- When the beacon is powered up, a series of information messages should be displayed in a serial-terminal (such as PuTTY) window along the lines of…

```
SEATRAC X-SERIES BEACON
Copyright (c) 2014 Blueprint Design Engineering Ltd. All rights reserved.
For further information, visit http://www.blueprintsubsea.com
Firmware Part BP00913, Version 1.0, Build 1915

Device Information...
Hardware Part Number = BP00795
Hardware Part Revision = 1
Hardware Serial Number = XXXXXX
Hardware Flags = 0x00000000
Hardware EUI48 = XX:XX:XX:XX:XX:XX
Bootloader Valid = 1
Bootloader Part Number = BP00912
Bootloader Version = v1.0.361
Bootloader Checksum = 0xBFC5FAB7
Application Valid = 1
Application Part Number = BP00913
Application Version = v1.0.1915
Application Checksum = 0x4DBF60E9
Pressure Sensor Id = 20905984
Pressure Sensor Type = PA
Pressure Sensor Calibrated = 2014-03-01
Pressure Sensor Min = 0.0 bar
Pressure Sensor Max = 200.0 bar

Ready...
```

- At this point the beacon is ready to receive serial commands. Type “#0281C1” and press return – this will send a CID_SYS_INFO command requesting hardware and firmware information.
- Something similar to the following information string should be returned (if not a formatting error message/code will be returned – see CST_E for appropriate return codes)…

```
$0234000000011B0301690E000000000000FF900301006901B7FAC5BFFF910301007A077504
63A973BA
```

- The fields in the above message can then be decoded using the information given in the CID_SYS_INFO command reference section of this document. Alternately, you could use the SeaTrac Tools application to send the above command, and use the "Journal" to decode the information fields as…

```
Received Len = 41 bytes (in 83 chars)
Received Data =
"$0234000000011B0301690E000000000000FF900301006901B7FAC5BFFF910301007A07750
463A973BA"
CmdId = SYS_INFO (0x02)
Uptime = 0x00000034 (52)
Section = 0x01 (1)
HardwarePartNumber = 0x031B (795)
HardwarePartRevision = 0x01 (1)
HardwareSerialNumber = 0x00000E69 (3689)
HardwareFlags = 0x00000000 (0)
BootValid = TRUE
BootPartNumber = 0x0390 (912)
BootVersionMajor = 0x01 (1)
BootVersionMinor = 0x00 (0)
BootVersionBuild = 0x0169 (361)
BootChecksum = 0xBFC5FAB7 (3217423031)
AppValid = TRUE
AppPartNumber = 0x0391 (913)
AppVersionMajor = 0x01 (1)
AppVersionMinor = 0x00 (0)
AppVersionBuild = 0x077A (1914)
AppChecksum = 0xA9630475 (2841838709)
```

## 4.2. Fetching, Updating and Storing Settings

Using a process similar to that described in section 4.1 above, you can make you application change the beacon settings using the following sequence of commands…

- First define the SETTINGS_T structure in your application – you will require this to read settings into.
- Send a CID_SETTINGS_GET command to the beacon, and parse the response into a field of type SETTINGS_T – this will include the current sensor calibration values in use.
- Modify the settings structure as required then write values back into the Beacons RAM using the CID_SETTINGS_SET command. Most settings will be applied immediately, but some (such as communication settings) will only be applied following a power-up or software reboot command.
- It may be required to store these new settings back into EEPROM, in which case now issue the CID_SETTINGS_SAVE command. Note that if automatic calculation of VOS and pressure offset are enabled, the previously specified values will be overwritten with the computed values.
- If a reboot if required to apply communication settings, issue a CID_SYS_REBOOT command now.

## 4.3. Requesting Status Output

Using a process similar to that described in section 4.1 above, you can make you application manually request the current beacon status via a Status Output Message using the following sequence of commands…

- In the terminal application, typing "#10000DC0" will prompt for a CID_STATUS message using the output flags configured in the settings.
- Assuming the default flags of "Environment", "AHRS" and "Magnetic Calibration" are specified, something similar to the following message will be received…

```
$10078D48100000000000B930C2000800000000000000480DE3FD0DFD320303FF2B0400005E
F273
```

- The above decodes as…

```
Received Len = 39 bytes (in 79 chars)
Received Data =
"$10078D48100000000000B930C2000800000000000000480DE3FD0DFD320303FF2B0400005
EF273"
CmdId = STATUS_STATE (0x10)
OutputFlags = 0x07 (7)
Timestamp = 0x000000000010488D (1067149)
EnvironmentSupply = 0x30B9 (12473)
EnvironmentTemperature = 194
EnvironmentPressure = 8
EnvironmentDepth = 0
EnvironmentVos = 0x0D48 (3400)
AttitudeYaw = -541
AttitudePitch = -755
AttitudeRoll = 818
MagCalBuf = 0x03 (3)
MagCalValid = TRUE
MagCalAge = 0x0000042B (1067)
MagCalFit = 0x5E (94)
```

- Alternately, you can specify a CID_STATUS with the first parameter as a combination of flags defined by the STATUS_BITS_T type – this allows the content of the message to be specified on a case-by-case basis.
- Automatic generation of Status Messages can be specified (or disabled) by choosing the appropriate value from the STATUSMODE_E enumeration and send this via either the CID_SETTINGS_SET or CID_STATUS_CFG_SET commands.

## 4.4. Acoustically Pinging Beacons

There are many different ways that Beacons can be made to acoustically interact with each other depending on the developer's operational requirements.

This section covers the use of basic interrogation and operating a simple network that tracks the position of several beacons using the PING protocol. Information on other protocols can be found in section 8 from page 125 onwards.

Using a process similar to that described in section 4.1 above, a single beacon can be Pinged using the following commands…

### Configuring Remote Beacons (to be interrogated)

For each beacon that will be deployed and Pinged, you may wish to apply/check the following settings prior to use.

- Start as described previously in section 4.2 by retrieving the beacon settings into a SETTINGS_T structure.
- Check that each remote beacon is assigned a unique beacon ID for the network addressing in the XCVR_BEACON_ID field.
- Check each remote beacon uses the same response turnaround time value in the XCVR_RESP_TIME field – typically this should be 10ms.
- Program any settings changed back into the beacon using the CID_SETTINGS_SET and CID_SETTINGS_SAVE commands.

### Configuring the Local Beacon (that performs the interrogation)

- Start as described previously in section 4.2 by retrieving the beacon settings into a SETTINGS_T structure.
- Request that position fix messages are generated by the acoustic transponder by settings the XCVR_FIX_MSGS flag in the XCVR_FLAGS field. Additionally specifying the XCVR_DIAG_MSGS flag will also request that further transceiver activity and status information is output.
- Setup the range (and hence amount of time) the transponder will wait for a remote beacon to respond to a request before a timeout error is raised by specifying the value of the XCVR_RANGE_TMO value.
- Check that the beacon is assigned a unique beacon ID for the network addressing in the XCVR_BEACON_ID field.
- Check the beacon uses the same response turnaround time value as the remote beacons, in the XCVR_RESP_TIME field – typically this should be 10ms.
- Program any settings changed back into the beacon using the CID_SETTINGS_SET and CID_SETTINGS_SAVE commands.

### Interrogating Beacons

- When the acoustic transponder is idle, send the CID_PING_SEND command along with a remote Beacon ID code to start a ping sequence. The beacon will immediately output a status response message indicating if the transponder is busy or if transmission has started. The acoustic transceiver will transmit a PING Request message
- On reception of the PING Request, the remote beacon will output as CID_PING_REQ serial message locally – this is for information only, so it does not matter if the remote beacon has a serial link connected.
- At some point later in time (proportional to the range of the remote beacon) as PING Response acoustic message should be received and the local beacon will signal success with a CID_PING_RESP message.

- Because of the previous configuration, the local beacon will generate a CID_XCVR_FIX message that contains ranging information. Additionally, if the local beacon is an X150 model this message will also contain positioning information.
- If no response is received by the local beacon, after the previously configured Range Timeout period has been observed a CID_PING_ERROR message will be generated indicating the beacon has timed out.

### Controlling Algorithm

By using the above procedure, developers can build their own control algorithms that poll a network of remote beacons, obtaining positing information for each one.

Iterations of the main loop should poll each beacon in turn (with a CID_PING_SEND command) and either wait for a CID_PING_RESP message (and CID_XCVR_FIX) to indicate success and position, or a CID_PING_ERROR message to indicate a timeout and the remote beacon cannot be communicated with (or is not present).
This information can be presented to the user in the most suitable way for the developers application.

## 4.5. Calibrating the AHRS

Using a process similar to that described in section 4.1 above, if desired you can implement your own AHRS calibration controls using the following sequence of commands…

### Accelerometer Calibration

- Prompt the user to hold the beacon steady and in an upright position and indicate when they are ready to commence.
- To give user feedback, turn on the outputting of accelerometer sensor and calibration status information using either the CID_SETTINGS_SET or CID_STATUS_CFG_SET command. It is recommend that the output rate be chosen to be either 5Hz or 10Hz with the STATUSMODE_E enumeration, and the STATUS_BITS_T type for STATUS_OUTPUT contain the ACC_CAL and AHRS_RAW_DATA bits.
- Send the CID_CAL_ACTION command with the ACTION parameter chosen to be CAL_ACC_RESET from the CAL_ACTION_E enumeration. This resets the measured limits that the accelerometer uses to determine the direction of Gravity (the vertical reference).
- As CID_STATUS messages are automatically generated, decode and show the Accelerometer limits values to the user. Prompt the user to slowly start moving the beacon around the vertical position to find the maximum Z value, then slowly invert the beacon to find the minimum Z value. The emphasis should be on slow and gentle movements to avoid acceleration peaks from being detected.
- Prompt the user to then hold the beacon horizontally and slowly rotate it clockwise and anti-clockwise to find the minimum and maximum values for the X and Y axis.
- The user should indicate when all 6 values have been successfully found, and the CID_CAL_ACTION command with the ACTION parameter chosen to be CAL_ACC_CALC (from the CAL_ACTION_E enumeration) to compute the Pitch and Roll limits.

- Optionally at this point the Accelerometer Calibration could be stored to EEPROM using the CID_SETTINGS_SAVE command.

### Magnetometer Calibration

- To give user feedback, turn on the outputting of magnetometer sensor and calibration status information using either the CID_SETTINGS_SET or CID_STATUS_CFG_SET command. It is recommend that the output rate be chosen to be either 5Hz or 10Hz with the STATUSMODE_E enumeration, and the STATUS_BITS_T type for STATUS_OUTPUT contain the MAG_CAL and AHRS_RAW_DATA bits.
- Send the CID_CAL_ACTION command with the ACTION parameter chosen to be CAL_MAG_RESET from the CAL_ACTION_E enumeration. This resets the magnetometer measurement buffer ready for a new set of data about the surrounding magnetic environment.
- Prompt the user to start rotating the beacon around all 3-axis in 3D space. As the beacon is rotated the Pitch and Roll information is used to build up a 3D magnetic map surrounding the beacon in the calibration buffer.
- Displaying the MAG_CAL_BUF value output in CID_STATUS message can give the user feedback on the calibration process. When the value reaches 100, calibration is complete and the user can be prompted to stop movement.
- Send the CID_CAL_ACTION command with the ACTION parameter chosen to be CAL_MAG_CALC (from the CAL_ACTION_E enumeration) to compute the Hard and Soft Iron compensation coefficients. The new calibration will be applied immediately.
- Optionally at this point the Magnetometer Calibration could be stored to EEPROM using the CID_SETTINGS_SAVE command.

# 5. Serial Interface Protocol

Communications between a SeaTrac Beacon and controlling device (such as a PC or embedded system) take the form of Command Messages sent to the Beacon, and Response (or Status) Messages received back from it.

All Command Messages issued to the Beacon are acknowledged by a Response Message, and occasionally Status messages are generated by the beacon in response to system events (such as message reception etc).

## 5.1. Serial Settings

Messages are sent over the RS232 serial link with the following settings…

- 115200 Baud Rate (the baud rate can be adjusted via the beacon settings – see the Beacon User Manual, or CID_SETTINGS_SET command for further details).
- 8 Data Bits
- No Parity
- 1 Start Bit
- 2 Stop Bits
- No handshaking / flow-control

## 5.2. Message Format

For ease of implementation and diagnostics, all messages are transmitted using ASCII encoded characters.

Command and Response messages share the same message structure, as shown in the diagrams below…

### Command Message (from PC to Beacon)

| **#** | **CID** | **Command Payload** | **CSUM** | **<CR><LF>** |
|---|---|---|---|---|
| *1 char* | *2 chars* | *multiple of 2 chars* | *4 chars* | *2 chars* |

### Response Message (from Beacon to PC)

| **$** | **CID** | **Command Response** | **CSUM** | **<CR><LF>** |
|---|---|---|---|---|
| *1 char* | *2 chars* | *multiple of 2 chars* | *4 chars* | *2 chars* |

## 5.3. Synchronisation Characters

To indicate the start of a message, a unique synchronisation character ('#' or '$') is used, whose presence is guaranteed not to occur anywhere else other than at the start of a valid message.

- Messages from the PC to the Beacon should start with a '#' character (ASCII code 35).
- Messages from the Beacon to the PC start with a '$' character (ASCII code 36).

## 5.4. Data Encoding

Following the synchronisation character, all message data is encoded in ASCII as hexadecimal character pairs describing bytes of data – where the first received character represents the most-significant nibble (bits 7 to 4) and the second character represent the lest-significant nibble (bits 3 to 0).
This means that only hexadecimal characters '0' to '9' and 'A' to 'F' are valid during decoding of the message content following the synchronisation character. All other characters are invalid.

For data values that require more than one byte in size to transmit (i.e. 16-bit Words and 32-bit DWords), the value is transmitted least-significant-byte first (Little-Endian).

Data lengths should always be a multiple of 2 as single nibble (character) values are never used.

For example receiving characters…

- Sequence 4A = 0x4A = 0b010 01010 = 74 decimal
- Sequence 1234 = 0x3412 = 0b0011 0100 0001 0010 = 13330 decimal

The end of the message is indicated by the 'Carriage-Return/Line-Feed' character pair (<CR><LF>) of ASCII values 13 (0x0D) and 10 (0x0A) respectively, allowing data to be displayed on a variety of terminal applications.

## 5.5. Command ID Codes

Following the synchronisation character, all messages start with a single byte (2 character) Command Identification Code (CID), indicating the purpose of the following data payload.

All acknowledgement Response Messages sent from the Beacon use the same CID as that specified in the Command Message sent to the Beacon.

For a list of valid CID codes see section 6.3.6, and section 7 for definitions of their purpose.

## 5.6. Checksums

The last 4 characters of each message (prior to the <CR><LF> characters) represent the 16-bit checksum value (send in Little-Endian form).

Checksums are computed by processing in a sequential byte-wise fashion the messages **binary content** after (but not including) the synchronisation character, and not including the checksum value itself – i.e. the CID and Payload message fields.
This means that the ASCII-Hex characters of the message string should first be decoded into an array of bytes representing their values, as described in section 5.4 (such that "32A9C4" would become 0x32, 0xA9, 0xC4), and then the array values used to compute the checksum.
This applies both to received and transmitted messages.

Message checksums are generated using the CRC-16-IBM polynomial of $x^{16} + x^{15} + x^2 + 1$. (Normal polynomial 0x8005, Reversed polynomial of 0xA001)

When receiving a message, the computed checksum should be compared against the received checksum, and if the two values match then the message is considered valid.

When transmitting a message, a buffer should be populated with the CID and message payload, then the checksum of this computed and appended to the end of the buffer.

The following C code shows a simple algorithm for computing the CRC of the message buffer…

```
/*!
Function that computes the CRC16 value for an array of sequential bytes
stored in memory.
NB: Types uint8 and uint16 represent unsigned 8 and 16 bit integers
Respectively.
@param buf   Pointer to the start of the buffer to compute the CRC16 for.
@param len   The number of bytes in the buffer to compute the CRC16 for.
@result      The new CRC16 value.
*/
uint16 CalcCRC16(uint8* buf, uint16 len)
{
	uint16 poly = 0xA001;
	uint16 crc = 0;

	for(uint16 b = 0; b < len; b++) {
		uint8 v = *buf;
		for(uint8 i = 0; i < 8; i++) {
			if((v & 0x01) ^ (crc & 0x01)) {
				crc >>= 1;
				crc ^= poly;
			}
			else {
				crc >>= 1;
```

```
                }
                v >>= 1;
            }
            buf++;
        }
        return crc;
}
```

The following message strings show typical synchronisation, CID and checksum data, and can be used to validate checksum algorithm implementations…

- `#0281C1` — CID = 0x02, Checksum = 0xC181
- `#15C1CF` — CID = 0x15, Checksum = 0xCFC1
- `#10000DC0` — CID = 0x10, Payload = 0x00, Checksum = 0xC00D
- `#4002B001` — CID = 0x40, Payload = 0x02, Checksum = 0x01B0
- `$31020104000000001109` — CID = 0x32, Payload = 7 bytes, Checksum = 0x0911
- `$0282330000011B030169 0E000000000000FF9003 01006901B7FAC5BFFF91 0301007A07750463A95DDE` — CID = 0x02, Payload = 40 bytes, Checksum = 0xDE5D

# 6. Message Field Type and Constant Definitions

Message payloads contain a sequence of data fields as defined by the CID code at the start of the message. Depending on the CID code specified, the data fields may differ in length and meaning, but all are built up out of simple defined data types discussed in this section…

## 6.1. Primitive Types

For basic exchange of data, the following primitive numeric data types are used. Unless otherwise stated all multi-byte data types are transmitted least-significant-byte first (Little-Endian).

| | |
|---|---|
| **BOOLEAN** | A Boolean flag encoded in a byte (UINT8) with 0x00 representing FALSE, and a non-zero value (typically 0xFF) representing TRUE. |
| **DOUBLE**[1] | An eight byte representation of a Double-precision floating-point number as defined by the IEEE754 standard (1 sign bit, 11 bit exponent, 52 bit fraction). |
| **FLOAT**[2] **(or SINGLE)** | An four byte representation of a Single-precision floating-point number as defined by the IEEE754 standard (1 sign bit, 8 bit exponent, 23 bit fraction). |
| **INT8** | A single byte representing an 8-bit signed integer (value between -128 and 127) |
| **INT16** | A two byte representation of a 16-bit signed integer (value between -32768 and 32767) |
| **INT32** | A four byte representation of a 32-bit signed integer (value between -2147483648 and 2147483647) |
| **INT64** | An eight byte representation of a 64-bit signed integer (value between -9223372036854775808 and 9223372036854775807) |
| **UINT8** | A single byte representing an 8-bit unsigned integer (value between 0 and 255) |
| **UINT16** | A two byte representation of a 16-bit unsigned integer (value between 0 and 65535) |
| **UINT32** | A four byte representation of a 32-bit unsigned integer (value between 0 and 4294967295) |
| **UINT64** | An eight byte representation of a 64-bit unsigned integer (value between 0 and 18446744073709551615) |

[1] For further details, see http://en.wikipedia.org/wiki/Double-precision_floating-point_format.
[2] For further details, see http://en.wikipedia.org/wiki/Single_precision.

## 6.2. Arrays

Arrays define sequential and continuous (packed) allocations of memory for a specific data type.
In the scope of this documentation, arrays are defined by the use of square brackets ("[ x ]") after a data type, with the value enclosed within the brackets denoting the number of storage elements required. Examples are…

- UINT8[12] – Defines an array of 12 UINT8 elements.
- DOUBLE[6] – Defines an array of 6 DOUBLE elements (48 bytes in total – 6 × 8 bytes).

Unless otherwise stated, arrays element indices are specified from 0 up to the number of array elements minus 1 – i.e. for a definition of x = UINT8[12], elements are defined as x[0] to x[11].

## 6.3. Enumerations & Constants

Enumerated types (or Enums) are data types that have a pre-defined set of valid values (sometimes called enumerators), and whose value can be set to store one of these.

If a value is stored that is not a defined enumerator, then the value should be considered invalid and unexpected application behaviour may occur as a result.

Depending on programming style, enumerations may also be defined as a storage value, such as a UINT8 and value enumerators defined as constants that can then be assigned to this value.



Unless stated otherwise, all enumerations defined below are stored and transmitted as UINT8 values.

### 6.3.1. AMSGTYPE_E (Acoustic Message Type)

The AMSGTYPE_E enumeration is used to specify a type of acoustic message, and determines how the message is processed and which responses are generated from beacons.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x0** | MSG_OWAY | Indicates an acoustic message is sent One-Way, and does not require a response. One-Way messages may also be broadcast to all beacons if required.<br><br>No USBL information is sent. |
| **0x1** | MSG_OWAYU | Indicates an acoustic message is sent One-Way, and does not require a response. One-Way messages may also be broadcast to all beacons if required.<br><br>Additionally, the message is sent with USBL acoustic information allowing an incoming bearing to be determined by USBL receivers, although range cannot be provided. |
| **0x2** | MSG_REQ | Indicates an acoustic message is sent as a Request type. This requires the receiver to generate and return a Response (MSG_RESP) message.<br><br>No USBL information is requested. |
| **0x3** | MSG_RESP | Indicate an acoustic message is sent as a Response to a previous Request message (MSG_REQ).<br><br>No USBL information is returned. |
| **0x4** | MSG_REQU | Indicates an acoustic message is sent as a Request type. This requires the receiver to generate and return a Response (MSG_RESP) message.<br><br>Additionally, the Response message should be returned with USBL acoustic information allowing a position fix to be computed by USBL receivers through the range and incoming signal angle. |
| **0x5** | MSG_RESPU | Indicate an acoustic message is sent as a Response to a previous Request message (MSG_REQ). |

| | | |
|---|---|---|
| | | Additionally, the message is sent with USBL acoustic information allowing the position of the sender to be determined through the range and incoming signal angle. |
| **0x6** | MSG_REQX | Indicates an acoustic message is sent as a Request type. This requires the receiver to generate and return a Response (MSG_RESP) message.<br><br>Additionally, the Response message should be returned with extended Depth and USBL acoustic information allowing a more accurate position fix to be computed by USBL receivers through the range, remote depth and incoming signal angle. |
| **0x7** | MSG_RESPX | Indicate an acoustic message is sent as a Response to a previous Request message (MSG_REQ).<br><br>Additionally, the message is sent with extended depth and USBL acoustic information allowing a more accurate position of the sender to be determined through the range, remote depth and incoming signal angle. |
| **0xFF** | MSG_UNKNOWN | This value is NEVER used to specify a message type when sending Acoustic Messages. However, on occasions certain structures need to specify “No Message Type” (for example see ACOFIX_T), and this value is used as an OUTPUT ONLY to indicate this. |

### 6.3.2. APAYLOAD_E (Acoustic Payload Identifier)

The APAYLOAD_E enumeration is used to specify how the payload contents of acoustic messages (see ACOMSG_T structures) are decoded and processed.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x0** | PLOAD_PING | Specified an acoustic message payload should be interpreted by the PING protocol handler.<br><br>PING messages provide the simplest (and quickest) method of validating the presence of a beacon, and determining its position. |
| **0x1** | PLOAD_ECHO | Specified an acoustic message payload should be interpreted by the ECHO protocol handler.<br><br>ECHO messages allow the function and reliability of a beacon to be tested, by requesting the payload contents of the message be returned back to the sender. |
| **0x2** | PLOAD_NAV | Specified an acoustic message payload should be interpreted by the NAV (Navigation) protocol handler.<br><br>NAV messages allow tracking and navigation systems to be built that use enhanced positioning and allow remote parameters of beacons (such as heading, attitude, water temperature etc) to be queried. |
| **0x3** | PLOAD_DAT | Specified an acoustic message payload should be interpreted by the DAT (Datagram) protocol handler.<br><br>DAT messages for the simplest method of data exchange between beacons, and provide a method of acknowledging data reception. |
| **0x4** | PLOAD_DEX | Specified an acoustic message payload should be interpreted by the DEX (Data Exchange) protocol handler.<br><br>DEX messages implement a protocol that allows robust bi-directional socket based data exchange with timeouts, acknowledgments and retry schemes. |

| | | |
|---|---|---|
| **0x5** | PLOAD_CFG | Specified an acoustic message payload should be interpreted by the CFG (Acoustic Config) protocol handler. |
| **0x6** | PLOAD_PINGER | Specified an acoustic message payload should be interpreted by the PINGER protocol handler. |
| **0x7 – 0xF** | | Reserved for future use. |
| **0X10** | PLOAD_TRACK | Specified an acoustic message payload should be interpreted by the TRACK protocol handler. |
| **0x11** | PLOAD_INTERRUPT | Specified an acoustic message payload should be interpreted by the INTERRUPT protocol handler. |

### 6.3.3. BAUDRATE_E (Serial Port Baud Rate)

The baud rate enumeration defines codes representing the speed of serial communications ports. Values specified outside those defined in the table below will default to BAUD_115200.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x07** | BAUD_4800 | 4800 bits per second |
| **0x08** | BAUD_9600 | 9600 bits per second |
| **0x09** | BAUD_14400 | 14400 bits per second |
| **0x0A** | BAUD_19200 | 19200 bits per second |
| **0x0B** | BAUD_38400 | 38400 bits per second |
| **0x0C** | BAUD_57600 | 57600 bits per second |
| **0x0D** | BAUD_115200 | 115200 bits per second |

### 6.3.4. BID_E (Beacon Identification Code)

Beacon Identification (BID) Codes are used to identify a specific beacon that should receive acoustic messages, or identify which beacon was the source (sender) of a message. Valid values are in the range from 0 to 100 and are typically sent and stored as a UINT8.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|

| | | |
|---|---|---|
| **0x0** | BEACON_ALL (for transmit)<br>RESERVED (for receive) | When used as an address for sending acoustic messages to, the value of 0x00 indicates “broadcast to all”.<br>When used as an identifier of a sender of a message, the value of 0x00 should be interpreted as unknown or invalid. |
| **0x1 to 0x64** | BEACON_ID_x | Values from 1 to 100 represent valid beacon identification codes. |

### 6.3.5. CAL_ACTION_E (Calibration Actions)

The Calibration Action enumeration defines what operation the beacon should perform when a CID_CAL_ACTION command is issued. Valid operations are…

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x00** | CAL_ACC_DEFAULTS | Sets the current accelerometer calibration coefficient values back to defaults. |
| **0x01** | CAL_ACC_RESET | Resets the accelerometer Min and Max filtered limit values measured by the sensor (does not modify the current calibration). |
| **0x02** | CAL_ACC_CALC | Calculates the new accelerometer calibration coefficients from the measured sensor limit values. |
| **0x03** | CAL_MAG_DEFAULTS | Sets the current magnetometer calibration values back to defaults for Hard and Sort Iron (does not clear the magnetic buffer). |
| **0x04** | CAL_MAG_RESET | Clears the magnetic calibration buffer (does not modify the current calibration). |
| **0x05** | CAL_MAG_CALC | Calculate and apply a new Magnetometer calibration from current magnetic buffer values. |
| **0x06** | CAL_PRES_OFFSET_RESET | Reset the pressure offset back to zero Bar. |
| **0x07** | CAL_PRES_OFFSET_CALC | Sets the pressure offset from the current measured pressure, zeroing the depth sensor reading. |

### 6.3.6. CID_E (Command Identification Codes)

Command Identification (CID) Codes are an enumeration (or defined set of constants) stored/transmitted in a UINT8 type at the start of Command and Response messages after the synchronisation character, with the purpose of identifying the message function and its payload.

The usage of each CID, including definition of message fields for both Command and Response payloads is discussed in section 7 from page 80.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **System Messages** | | |
| **0x01** | CID_SYS_ALIVE | Command sent to receive a simple alive message from the beacon. |
| **0x02** | CID_SYS_INFO | Command sent to receive hardware & firmware identification information. |
| **0x03** | CID_SYS_REBOOT | Command sent to soft reboot the beacon. |
| **0x04** | CID_SYS_ENGINEERING | Command sent to perform engineering actions. |
| **Firmware Programming Messages** | | |
| **0x0D** | CID_PROG_INIT | Command sent to initialise a firmware programming sequence. |
| **0x0E** | CID_PROG_BLOCK | Command sent to transfer a firmware programming block. |
| **0x0F** | CID_PROG_UPDATE | Command sent to update the firmware once program transfer has completed. |
| **Status Messages** | | |
| **0x10** | CID_STATUS | Command sent to request the current system status (AHRS, Depth, Temp, etc). |
| **0x11** | CID_STATUS_CFG_GET | Command sent to retrieve the configuration of the status system (message content and auto-output interval). |
| **0x12** | CID_STATUS_CFG_SET | Command sent to set the configuration of the status system (message content and auto-output interval). |
| **Settings Messages** | | |
| **0x15** | CID_SETTINGS_GET | Command sent to retrieve the working settings in use on the beacon. |

| | | |
|---|---|---|
| **0x16** | CID_SETTINGS_SET | Command sent to set the working settings and apply them. They are NOT saved to permanent memory until CID_ SETTINGS_SAVE is issued.<br>The device will need to be rebooted after this to apply some of the changes. |
| **0x17** | CID_SETTINGS_LOAD | Command sent to load the working settings from permanent storage and apply them.<br>Not all settings can be loaded and applied as they only affect the device on start-up. |
| **0x18** | CID_SETTINGS_SAVE | Command sent to save the working settings into permanent storage. |
| **0x19** | CID_SETTINGS_RESET | Command sent to restore the working settings to defaults, store them into permanent memory and apply them. |

### Calibration Messages

| | | |
|---|---|---|
| **0x20** | CID_CAL_ACTION | Command sent to perform specific calibration actions. |
| **0x21** | CID_AHRS_CAL_GET | Command sent to retrieve the current AHRS calibration. |
| **0x22** | CID_AHRS_CAL_SET | Command sent to set the contents of the current AHRS calibration (and store to memory)/ |

### Acoustic Transceiver Messages

| | | |
|---|---|---|
| **0x30** | CID_XCVR_ANALYSE | Command sent to instruct the receiver to perform a noise analysis and report the results. |
| **0x31** | CID_XCVR_TX_MSG | Message sent when the transceiver transmits a message. |
| **0x32** | CID_XCVR_RX_ERR | Message sent when the transceiver receiver encounters an error. |
| **0x33** | CID_XCVR_RX_MSG | Message sent when the transceiver receives a message (not requiring a response). |
| **0x34** | CID_XCVR_RX_REQ | Message sent when the transceiver receives a request (requiring a response). |
| **0x35** | CID_XCVR_RX_RESP | Message sent when the transceiver receives a response (to a transmitted request). |
| **0x37** | CID_XCVR_RX_UNHANDLED | Message sent when a message has been received but not handled by the protocol stack. |

| | | |
|---|---|---|
| **0x38** | CID_XCVR_USBL | Message sent when a USBL signal is decoded into an angular bearing. |
| **0x39** | CID_XCVR_FIX | Message sent when the transceiver gets a position/range fix on a beacon from a request/response. |
| **0x3A** | CID_XCVR_STATUS | Message sent to query the current transceiver state. |
| **0x3B** | CID_XCVR_TX_MSGCTRL_SET | Command send to set the state of the XCVR_TX_MSGCTRL bits in the XCVR_FLAGS settings register, and control the behaviour of the Transceiver Transmitter. |
| **0x3C** | CID_XCVR_USBL_TIME | Message sent out of an X150 beacon after a USBL signal has been decoded. The message indicates when the USBL signal was received. |
| **0x3D** | CID_XCVR_BASELINES | Message sent out of an X150 beacon after a USBL signal has been decoded. The message is only sent if the XCVR_BASELINES_MSGS flag is set in the beacon settings. |

### PING Protocol Messages

| | | |
|---|---|---|
| **0x40** | CID_PING_SEND | Command sent to transmit a PING message. |
| **0x41** | CID_PING_REQ | Message sent when a PING request is received. |
| **0x42** | CID_PING_RESP | Message sent when a PING response is received. |
| **0x43** | CID_PING_ERROR | Message sent when a PING response error/timeout occurs. |

### ECHO Protocol Messages

| | | |
|---|---|---|
| **0x48** | CID_ECHO_SEND | Command sent to transmit an ECHO message. |
| **0x49** | CID_ECHO_REQ | Message sent when an ECHO request is received. |
| **0x4A** | CID_ECHO_RESP | Message sent when an ECHO response is received with the echo response data. |
| **0x4B** | CID_ECHO_ERROR | Message sent when an ECHO response error/timeout occurs. |

### NAV Protocol Messages

| | | |
|---|---|---|
| **0x50** | CID_NAV_QUERY_SEND | Message sent to query navigation information from a remote beacon. |
| **0x51** | CID_NAV_QUERY_REQ | Message sent from a beacon that receives a NAV_QUERY. |

| | | |
|---|---|---|
| **0x52** | CID_NAV_QUERY_RESP | Message generated when the beacon received a response to a NAV_QUERY. |
| **0x53** | CID_NAV_ERROR | Message generated if there is a problem with a NAV_QUERY - i.e. timeout etc. |
| **0x58** | CID_NAV_QUEUE_SET | Message sent to set the contents of the packet data queue. |
| **0x59** | CID_NAV_QUEUE_CLR | Message sent to clear the contents of the packet data queue. |
| **0x5A** | CID_NAV_QUEUE_STATUS | Message sent to obtain the current status of the packet data queue. |
| **0x5B** | CID_NAV_STATUS_SEND | Message that is used to broadcast status information from one beacon (typically the USBL head) to others in the system. |
| **0x5C** | CID_NAV_STATUS_RECEIVE | Message generated when a beacon receives a NAV_STATUS message. |
| **0x5D** | CID_NAV_QUEUE_GET | Message to retrieve the contents of the packet data queue. |

### DAT Protocol Messages

| | | |
|---|---|---|
| **0x60** | CID_DAT_SEND | Message sent to transmit a datagram to another beacon |
| **0x61** | CID_DAT_RECEIVE | Message generated when a beacon receives a datagram. |
| **0x63** | CID_DAT_ERROR | Message generated when a beacon response error/timeout occurs for ACKs. |
| **0x64** | CID_DAT_QUEUE_SET | Message sent to set the contents of the packet data queue. |
| **0x65** | CID_DAT_QUEUE_CLR | Message sent to clear the contents of the packet data queue. |
| **0x66** | CID_DAT_QUEUE_STATUS | Message sent to obtain the current status of the packet data queue. |
| **0x67** | CID_DAT_QUEUE_GET | Message sent to get the contents of the packet data queue. |

### CFG Protocol Messages

| | | |
|---|---|---|
| **0x80** | CID_CFG_BEACON_GET | Message sent to query the remove Beacon Transceiver settings, by serial number (rather than Beacon ID). |
| **0x81** | CID_CFG_BEACON_SET | Message sent to set the remove Beacon Transceiver settings, by serial number (rather than Beacon ID). |
| **0x82** | CID_CFG_BEACON_RESP | Message output when a remote beacon response is received containing the current Beacon Transceiver settings (following a CID_CFG_BEACON_GET or CID_CFG_BEACON_SET command). |
| **0x83** | CID_CFG_REQ_SEND | Message sent to set and/or get configuration on/from a remote beacon by serial number. |

| | | |
|---|---|---|
| **0x84** | CID_CFG_REQ_RECEIVE | Message output when an acoustic configuration request has been received that is targeted at an external device connected to the SeaTrac. |
| **0x85** | CID_CFG_RESP_SEND | Message sent to trigger a response to a CID_CFG_REQ_RECEIVE. |
| **0x86** | CID_CFG_RESP_RECEIVE | Message output when a response has been received to a previously transmitted CID_CFG_REQ_SEND. |
| **0x87** | CID_CFG_LOCAL_LOCK | Message sent to lock the acoustic config protocol for the locally connected beacon. |
| **0x88** | CID_CFG_LOCAL_UNLOCK | Message sent to unlock the acoustic config protocol for the locally connected beacon. |

### TRACK Protocol Messages

| | | |
|---|---|---|
| **0x90** | CID_TRACK_SEND | Message sent to transmit a TRACK message. |
| **0x91** | CID_TRACK_REQ | Message sent from a beacon when a TRACK request is received. |
| **0x92** | CID_TRACK_RESP | Message sent from a beacon when a TRACK response is received. |
| **0x93** | CID_TRACK_ERROR | Message sent from a beacon when a TRACK response error or timeout occurs. |

### PINGER Protocol Messages

| | | |
|---|---|---|
| **0xA0** | CID_PINGER_SEND | Message sent to generate a 37KHz CW, 10ms Pulse (experimental). |

### INTERRUPT Protocol Messages

| | | |
|---|---|---|
| **0xB0** | CID_INTERRUPT_SEND | Message sent to generate an interrupt in the system. |
| **0xB1** | CID_INTERRUPT_RECEIVE | Message output when a beacon receives an interrupt from another beacon in the system. |
| **0xB2** | CID_INTERRUPT_RESET_IDLE | Message output when a beacon's transceiver has reset to an idle state following either sending or receiving an interrupt. |
| **0xB3** | CID_INTERRUPT_UNLOCK | Message output when a beacon is allowed to transmit an interrupt itself following receiving an interrupt. |

### 6.3.7. CST_E (Command Status Codes)

Command Status (CST) Codes are an enumeration (or set of defined constants) that are commonly used in Response messages sent from the beacon to indicate if a command executed successfully, or if not, what type of error occurred. CST codes are always transmitted as a UINT8 type.

Different Response messages may only implement a subset of the constants below, as appropriate for their function. Further details about the status codes each CID response provides are discussed in section 7 from page 80.

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **General Status Codes** | | |
| **0x00** | CST_OK | Returned if a command or operation is completed successful without error. |
| **0x01** | CST_FAIL | Returned if a command or operation cannot be completed. |
| **0x03** | CST_EEPROM_ERROR | Returned if an error occurs while reading or writing EEPROM data. |
| **Command Processor Status Codes** | | |
| **0x04** | CST_CMD_PARAM_MISSING | Returned if a command message is given that does not have enough defined fields for the specified CID code. |
| **0x05** | CST_CMD_PARAM_INVALID | Returned if a data field in a message does not contain a valid or expected value. |
| **Firmware Programming Status Codes** | | |
| **0x0A** | CST_PROG_FLASH_ERROR | Returned if an error occurs while writing data into the processors flash memory. |
| **0x0B** | CST_PROG_FIRMWARE_ERROR | Returned if firmware cannot be programmed due to incorrect firmware credentials or signature. |
| **0x0C** | CST_PROG_SECTION_ERROR | Returned if the firmware cannot be programmed into the specified memory section. |
| **0x0D** | CST_PROG_LENGTH_ERROR | Returned if the firmware length is too large to fit into the specified memory section, or not what the current operation is expecting. |

| | | |
|---|---|---|
| **0x0E** | CST_PROG_DATA_ERROR | Returned if there is an error decoding data in a firmware block. |
| **0x0F** | CST_PROG_CHECKSUM_ERROR | Returned if the specified checksum for the firmware does not match the checksum computed prior to performing the update. |

### Acoustic Transceiver Status Codes

| | | |
|---|---|---|
| **0x30** | CST_XCVR_BUSY | Returned if the transceiver cannot perform a requested action as it is currently busy (i.e. transmitting a message). |
| **0x31** | CST_XCVR_ID_REJECTED | Returned if the received message did not match the specified transceiver ID (and wasn't a Sent-To-All), and the message has been rejected. |
| **0x32** | CST_XCVR_CSUM_ERROR | Returned if received acoustic message's checksum was invalid, and the message has been rejected. |
| **0x33** | CST_XCVR_LENGTH_ERROR | Returned if an error occurred with message framing, meaning the end of the message has not been received within the expected time. |
| **0x34** | CST_XCVR_RESP_TIMEOUT | Returned if the transceiver has sent a request message to a beacon, but no response has been returned within the allotted waiting period. |
| **0x35** | CST_XCVR_RESP_ERROR | Returned if the transceiver has send a request message to a beacon, but an error occurred while receiving the response. |
| **0x36** | CST_XCVR_RESP_WRONG | Returned if the transceiver has sent a request message to a beacon, but received an unexpected response from another beacon while waiting. |
| **0x37** | CST_XCVR_PLOAD_ERROR | Returned by protocol payload decoders, if the payload can't be parsed correctly. |
| **0x3A** | CST_XCVR_STATE_STOPPED | Indicates the transceiver is in a stopped state. |
| **0x3B** | CST_XCVR_STATE_IDLE | Indicates the transceiver is in an idle state waiting for reception or transmission to start. |
| **0x3C** | CST_XCVR_STATE_TX | Indicates the transceiver is in a transmitting states. |
| **0x3D** | CST_XCVR_STATE_REQ | Indicates the transceiver is in a requesting state, having transmitted a message and is waiting for a response to be received. |

| | | |
|---|---|---|
| **0x3E** | CST_XCVR_STATE_RX | Indicates the transceiver is in a receiving state. |
| **0x3F** | CST_XCVR_STATE_RESP | Indicates the transceiver is in a responding state, where a message is being composed and the “response time” period is being observed. |

DEX Protocol Status Codes

| | | |
|---|---|---|
| **0x70** | CST_DEX_SOCKET_ERROR | Returned by the DEX protocol handler if an error occurred trying to open, close or access a specified socket ID. |
| **0x71** | CST_DEX_RX_SYNC | Returned by the DEX protocol handler when receiver synchronisation has occurred with the socket master and data transfer is ready to commence. |
| **0x72** | CST_DEX_RX_DATA | Returned by the DEX protocol handler when data has been received through a socket. |
| **0x73** | CST_DEX_RX_SEQ_ERROR | Returned by the DEX protocol handler when data transfer synchronisation has been lost with the socket master. |
| **0x74** | CST_DEX_RX_MSG_ERROR | Returned by the DEX protocol handler to indicate an unexpected acoustic message type with the DEX protocol has been received and cannot be processed. |
| **0x75** | CST_DEX_REQ_ERROR | Returned by the DEX protocol handler to indicate a error has occurred while responding to a request (i.e. lack of data). |
| **0x76** | CST_DEX_RESP_TMO_ERROR | Returned by the DEX protocol handler to indicate a timeout has occurred while waiting for a response back from a remote beacon with requested data. |
| **0x77** | CST_DEX_RESP_MSG_ERROR | Returned by the DEX protocol handler to indicate an error has occurred while receiving response back from a remote beacon. |
| **0x78** | CST_DEX_RESP_REMOTE_ERROR | Returned by the DEX protocol handler to indicate the remote beacon has encountered an error and cannot return the requested data or perform the required operation. |

### CFG Protocol Status Codes

| | | |
|---|---|---|
| **0x80** | CST_CFG_RESP_NOT_ALLOWED | Returned by the CFG protocol handler if an attempt has been made to send a CID_CFG_RESP_SEND more than once after receiving a request, or if no request has been received at all. |
| **0x81** | CST_CFG_ALREADY_LOCKED | Returned by the CFG protocol handler if an attempt is made to lock a beacon using the CID_CFG_LOCAL_LOCK command when that beacon is already locked. |
| **0x82** | CST_CFG_NOT_LOCKED | Returned by the CFG protocol handler if an attempt is made to unlock a beacon using the CID_CFG_LOCAL_UNLOCK command when that beacon is not locked. |

### INTERRUPT Protocol Status Codes

| | | |
|---|---|---|
| **0xB0** | CST_INTERRUPT_LOCKED | Returned by the INTERRUPT protocol handler if an attempt has been made to send an interrupt tone and the protocol is in a locked state due to receiving an interrupt from another beacon. |

### 6.3.8. STATUSMODE_E (Status Output Mode)

The Status Mode enumeration is used to specify how often periodic status output messages are automatically generated…

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x0** | STATUS_MODE_MANUAL | Status output message are not generated automatically, only upon manual request by sending the CID_STATUS command. |
| **0x1** | STATUS_MODE_1HZ | Status output message are not generated at 1 second (1Hz) intervals. |
| **0x2** | STATUS_MODE_2HZ5 | Status output message are not generated at 0.4 second (2.5Hz) intervals. |
| **0x3** | STATUS_MODE_5HZ | Status output message are not generated at 0.2 second (5Hz) intervals. |
| **0x4** | STATUS_MODE_10HZ | Status output message are not generated at 0.1 second (10Hz) intervals. |
| **0x5** | STATUS_MODE_25HZ | Status output message are not generated at 0.04 second (25Hz) intervals.<br><br>Be wary using this setting, as on slow communication baud rates, this setting may lead to a situation where more data is required to be sent down the link in a set period of time than the link is physically capable of sending – and the beacon processor may stall. |

### 6.3.9. XCVR_TXMSGCTRL_E (Transmit Message Control)

Controls the Transmit behaviour of the Acoustic Transceiver engine…

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x0** | XCVR_TXMSG_ALLOW_ALL | Allow transmission of all messages |
| **0x1** | XCVR_TXMSG_BLOCK_RESP | Block transmission of response messages. Automatic transmission of Response Messages after a Request message is received are blocked. |
| **0x2** | Not Used | |

| | | |
|---|---|---|
| **0x3** | XCVR_TXMSG_BLOCK_ALL | Block transmission of all messages. Transmit commands function as normal except no physical transmission is started. |

### 6.3.10. CFG_ADDRESS_E (Acoustic Config Address)

Addresses for the available configuration parameters when using version 2 of the CFG protocol. This table includes a column detailing the data format for each config address.

| *Value* | *Symbolic Name* | *Summary* | *Data* |
|---|---|---|---|
| **0x0** | CFG_ADDRESS_GET_FW_VERSION | Get main app FW version | UINT8[3] (X.Y.Z) |
| **0x1** | CFG_ADDRESS_GET_BEACON_TYPE | Get the beacon type | UINT8 (BEACON_TYPE_E) |
| **0x2** | CFG_ADDRESS_GET_BEACON_ID | Get the beacon ID | UINT8 (BID_E) |
| **0x3** | CFG_ADDRESS_SET_BEACON_ID | Set the beacon ID | UINT8 (BID_E) |
| **0x4** | CFG_ADDRESS_GET_TURNAROUND | Get the response turnaround time | UINT16 (in milliseconds) |
| **0x5** | CFG_ADDRESS_SET_TURNAROUND | Set the response turnaround time | UINT16 (in milliseconds) |
| **0x6** | CFG_ADDRESS_GET_RANGE_TIMEOUT | Get the range timeout | UINT16 (in metres) |
| **0x7** | CFG_ADDRESS_SET_RANGE_TIMEOUT | Set the range timeout | UINT16 (in metres) |
| **0x8** | CFG_ADDRESS_SET_LOCATOR_ENABLE | Sets the enabled state of locator beacon mode | BOOLEAN |
| **0x9** | CFG_ADDRESS_SET_TXMSGCTRL | Sets the TX message control value | UINT8 (XCVR_TXMSGCTRL_E) |
| **…** | | | |
| **0xFE** | CFG_ADDRESS_CFG_LOCK | Lock the CFG protocol until an unlock (0xFF) is received with a matching PIN. | UINT16 (PIN) |
| **0xFF** | CFG_ADDRESS_CFG_UNLOCK | Unlock the CFG protocol | UINT16 (PIN) |

### 6.3.11. BEACON_TYPE_E (Beacon Type)

Valid values for beacon type fields:

| *Value* | *Symbolic Name* | *Summary* |
|---|---|---|
| **0x0** | UNKNOWN | Beacon type is unknown |
| **0x1** | X010 | An X010 beacon |
| **0x2** | X110 | An X110 beacon |
| **0x3** | X150 | An X150 USBL beacon |
| **0x4** | D010 | A D010 diver beacon |

## Structures

Structures (sometimes called Records or Structs) are declarations defining complex data types of physically grouped variables placed under one name in a block of memory.
For the scope of this document, the members (or fields) of Structures should be assumed to be defined sequentially in memory, with no additional packing bytes added.

### 6.3.12. ACOMSG_T (Acoustic Message)

The Acoustic Message structure described the contents of packets that are transmitted between beacons acoustically, and form the basis for all information transfer in higher level protocols.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_DEST_ID | BID_E | Identifier for the beacon the message is or was the recipient of the message. |
| MSG_SRC_ID | BID_E | Identifier for the beacon that generated or sent the message. |
| MSG_TYPE | AMSGTYPE_E | Value that indicates the type of message being sent |
| MSG_DEPTH | UINT16 | Value that is only valid when MSG_TYPE is MSG_RESPX (an extended USBL response) and contains the depth sensor reading from the remote beacon. The depth is encoded 0.5m steps, and so should be divided by 2 to obtain a depth in metres. |
| MSG_PAYLOAD_ID | APAYLOAD_E | Value that indicates the type of payload the message contains. |
| MSG_PAYLOAD_LEN | UINT8 | Values specifying how many bytes in the payload array are valid and in use. Valid values are from 0 (for no payload) to 31. |
| RESPONSE_COUNT | UINT8 | Value that is only valid when MSG_PAYLOAD_ID is PLOAD_TRACK (a TRACK message) and contains the number of TRACK responses requested from the remote beacon. Valid values are from 1 to 16. |
| SHORT_RESP_INTERVAL | BOOL | Value that is only valid when MSG_PAYLOAD_ID is PLOAD_TRACK (a TRACK message) and specifies whether the XCVR_BASELINES_MSGS transceiver flag is enabled. |

| | | |
|---|---|---|
| MSG_PAYLOAD | UINT8[31] | Array of 31 bytes that contains the payload of the acoustic message.<br>Only the number of bytes specified in the PAYLOAD_LEN parameter are valid, while the contents of the other locations are undefined.<br>The exact function and definition of the PAYLOAD bytes depends on the type of payload specified in the PAYLOAD_ID parameter. |

### 6.3.13. ACOFIX_T (Acoustic Position and Range Fix Summary)

The Acoustic Fix structure is produced by the acoustic transceiver module and contains a summary of any information relating to a received signal – this includes current beacon depth, beacon attitude, water VOS, signal strength and any information that can be computed relating to the remote beacons range and position.

The data record varies depending on the contents of the FLAGS field, with required fields being appended to the end of the record as required.

For details of attitude definitions refer to section 9.1 on page 184.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| DEST_ID | BID_E | The ID code of the beacon that this data is sent to. Normally this is the local beacon ID code, but a value of BEACON_ALL indicates data has been transmitted to all beacons.<br>Valid values are form 0 to 100. |
| SRC_ID | BID_E | The ID code of the beacon that sent the data.<br>Valid values are form 1 to 100. |
| FLAGS | UINT8 | A bit-field of flags used to indicate what the rest of the record contains.<br>Bit values are… |
| • Bit[7:5] = RESERVED | | These bits are reserved for future use |
| • Bit[4] = POSITION_FLT_ERROR | | If this bit is true, it indicates the position filter has identified that the position specified in the fix may be invalid based on the beacons previous position, the define beacons motion limits and the time since last communication. However, the position fields still contain the USBL computed position and it is up to the user if they wish to reject this fix, or use it in some direct or weighted fashion. |
| • Bit[3] = POSITION_ENHANCED | | If this bit is set, it indicates the Position fix has been computed from an Enhanced USBL return – this means the Depth will be the value from the remote beacons depth sensor rather than computed form the incoming signal angle. |
| • Bit[2] = POSITION_VALID | | If this bit is set, it indicates the record contains the Position fields discussed below. |
| • Bit[1] = USBL_VALID | | If this bit is set, it indicates the record contains the USBL fields discussed below. |
| • Bit[0] = RANGE_VALID | | If this bit is set, it indicates the record contains the Range fields discussed below. |

| | | |
|---|---|---|
| MSG_TYPE | AMSGTYPE_E | The type of acoustic message received to generate this fix. |
| ATTITUDE_YAW | INT16 | The yaw angle (relative to magnetic north) of the local beacon when the fix was computed.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| ATTITUDE_PITCH | INT16 | The pitch angle of the local beacon when the fix was computed.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| ATTITUDE_ROLL | INT16 | The roll angle of the local beacon when the fix was computed.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| DEPTH_LOCAL | UINT16 | The reading from the local beacon depth sensor when the fix was calculated.<br>Values are encoded in decimetres, so divide by 10 to obtain a value in metres to a 0.1m resolution. |
| VOS | UINT16 | The velocity of sound value used for the computation of the remote beacon's range based on timing information.<br>Values are encoded in decimetres-per-second, so divide by 10 for a value in metres-per-second. |
| RSSI | INT16 | The Received Signal Strength Indicator value for the received message, encoded in centibels. To decode, divide this value by 10 for decibels to a 0.1 dB resolution. |

### Range Fields

If the message FLAGS parameter contains the RANGE_VALID bit, then the following fields are sequentially appended to the record…

| | | |
|---|---|---|
| RANGE_COUNT | UINT32 | The number of 16kHz timer intervals that were counted between Request message transmission and Response message reception. |
| RANGE_TIME | INT32 | The time in seconds derived from the RANGE_COUNT value, and with internal timing offsets and compensation applied.<br>Values are encoded in 100 nanosecond multiples, so divide by 10000000 to obtain a value in seconds. |

| | | |
|---|---|---|
| RANGE_DIST | UINT16 | The resolved line-of-sight distance to the remote beacon, based on the RANGE_TIME and VOS values.<br>Values are encoded in decimetres, so divide by 10 for a value in metres. |

### USBL Fields

If the message FLAGS parameter contains the USBL_VALID bit, then the following fields are sequentially appended to the record…

| | | |
|---|---|---|
| USBL_CHANNELS | UINT8 | The number of USBL receiver channels being used to compute the signal angle. Typically this value is either 3 or 4. |
| USBL_RSSI | INT16[x] | An array of the received signal strengths for each of the USBL receiver channels, where “x” is the value defined by the CHANNELS field.<br>Values are encoded in centi-Bels, so divide by 10 to obtain a value in decibels to a resolution of 0.1dB. |
| USBL_AZIMUTH | INT16 | The incoming signal azimuth angle from 0° to 360°.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| USBL_ELEVATION | INT16 | The incoming signal elevation angle from -90° to +90°.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| USBL_FIT_ERROR | INT16 | The fit error value returns a number that indicates the quality of fit (or confidence) of the signal azimuth and elevation values from the timing and phase-angle data available.<br><br>Values are dimensionless. Divide the value by 100 to obtain a signed floating-point value to a resolution of 0.01.<br><br>Smaller values towards 0.00 indicate a better fit, while larger values (increasing above 2.00-3.00) indicate poorer fits and larger error tolerances. |

### Position Fields

If the message FLAGS parameter contains the POSITION_VALID bit, then the following fields are sequentially appended to the record…

| | | |
|---|---|---|
| POSITION_EASTING | INT16 | The Easting distance component of the relative position of the remote beacon to the local beacon computed from the range, incoming signal angle, local beacon depth, attitude and magnetic heading.<br><br>Values are encoded in decimetres, so divide by 10 for a value in metres. |
| POSITION_NORTHING | INT16 | The Northing distance component of the relative position of the remote beacon to the local beacon computed from the range, incoming signal angle, local beacon depth, attitude and magnetic heading.<br><br>Values are encoded in decimetres, so divide by 10 for a value in metres. |
| POSITION_DEPTH | INT16 | The vertical Depth distance component of the remote beacon from the surface - computed from the range, incoming signal angle, local beacon depth, attitude and magnetic heading.<br><br>Values are encoded in decimetres, so divide by 10 for a value in metres.<br><br>NB: If the 'Fix' has been obtained by a MSG_REQU (Usbl) type request, then this value is computed from the beacon's attitude and incoming signal angle.<br>If a MSG_REQX (Enhanced) type request has been used, then this value is the remotely transmitted beacon depth sensor value. |

### 6.3.14. AHRSCAL_T (AHRS Calibration Coefficients)

An AHRS calibration structure contains all the coefficients required for the accelerometer, magnetometer and gyroscope sensors to produce valid yaw, pitch and roll attitude information.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| ACC_MIN_X | INT16 | The accelerometer X-axis sensor value that corresponds to -1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is -270. |
| ACC_MIN_Y | INT16 | The accelerometer Y-axis sensor value that corresponds to -1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is -270. |
| ACC_MIN_Z | INT16 | The accelerometer Z-axis sensor value that corresponds to -1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is -270. |
| ACC_MAX_X | INT16 | The accelerometer X-axis sensor value that corresponds to +1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is 270. |
| ACC_MAX_Y | INT16 | The accelerometer Y-axis sensor value that corresponds to +1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is 270. |
| ACC_MAX_Z | INT16 | The accelerometer Z-axis sensor value that corresponds to +1G of gravitational force.<br>Valid values lie in the range -1000 to +1000.<br>Default value is 270. |
| MAG_VALID | BOOLEAN | Flag is true when the calibration contains (or represents) a valid set of coefficients.<br>Writing an invalid calibration causes no compensation to be performed on sensor values.<br>Reading this flag as false indicates no dynamic calibration has been computed or loaded from EEPROM memory. |

| | | |
|---|---|---|
| MAG_HARD_X | FLOAT | The magnetometer X-axis sensor offset value to compensate for Hard Iron effects.<br>Valid values lie in the range -2000 to +2000.<br>Default value is 0. |
| MAG_HARD_Y | FLOAT | The magnetometer Y-axis sensor offset value to compensate for Hard Iron effects.<br>Valid values lie in the range -2000 to +2000.<br>Default value is 0. |
| MAG_HARD_Z | FLOAT | The magnetometer Z-axis sensor offset value to compensate for Hard Iron effects.<br>Valid values lie in the range -2000 to +2000.<br>Default value is 0. |
| MAG_SOFT_X | FLOAT | The magnetometer X-axis sensor scaling value to compensate for Soft Iron effects.<br>Valid values lie in the range -10 to +10.<br>Default value is 1. |
| MAG_SOFT_Y | FLOAT | The magnetometer Y-axis sensor scaling value to compensate for Soft Iron effects.<br>Valid values lie in the range -10 to +10.<br>Default value is 1. |
| MAG_SOFT_Z | FLOAT | The magnetometer Z-axis sensor scaling value to compensate for Soft Iron effects.<br>Valid values lie in the range -10 to +10.<br>Default value is 1. |
| MAG_FIELD | FLOAT | The normalised (not actual) magnetic field used for magnetometer calibration.<br>Valid values lie between 0 and 100, with a typical value for idea fit being 50.<br>Default value is 0. |
| MAG_ERROR | FLOAT | The fit error of the magnetic calibration.<br>Values are expressed as a percentage between 0 and 100.<br>Default value is 100 representing 100% error. |
| GYRO_OFFSET_X | INT16 | The rotational rate gyroscope X-axis sensor offset.<br>Valid values lie in the range -1000 to +1000.<br>Default value of 0. |
| GYRO_OFFSET_Y | INT16 | The rotational rate gyroscope Y-axis sensor offset.<br>Valid values lie in the range -1000 to +1000.<br>Default value of 0. |

| | | |
|---|---|---|
| GYRO_OFFSET_Z | INT16 | The rotational rate gyroscope Z-axis sensor offset.<br>Valid values lie in the range -1000 to +1000.<br>Default value of 0. |

### 6.3.15. FIRMWARE_T (Firmware Information)

The FIRMWARE_T structure is used to describe configuration information for the firmware that is currently loaded into memory.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| VALID | BOOLEAN | Flag when true indicating the firmware is valid and allowed to execute. |
| PART_NUMBER | UINT16 | The part number of the Bootloader firmware. |
| VERSION_MAJ | UINT8 | The major version number of the firmware (when expressed in the form `Version <major>.<minor>.<build>`). |
| VERSION_MIN | UINT8 | The minor version number of the firmware (when expressed in the form `Version <major>.<minor>.<build>`). |
| VERSION_BUILD | UINT16 | The sequentially assigned build number of the firmware (when expressed in the form `Version <major>.<minor>.<build>`). |
| CHECKSUM | UINT32 | The CRC32 checksum of the firmware. |

### 6.3.16. HARDWARE_T (Hardware Information)

The HARDWARE_T structure is used to describe information related to the hardware configuration of the beacon.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| PART_NUMBER | UINT16 | The hardware product part number, for example…<br>• 795 = SeaTrac X150 USBL Beacon<br>• 843 = SeaTrac X110 Modem Beacon |
| PART_REV | UINT8 | The hardware product part revision. |
| SERIAL_NUMBER | UINT32 | The unique serial number of the beacon. |
| FLAGS_SYS | UINT16 | Additional flags field defining factory set hardware capabilities and features.<br>Currently reads as 0, reserved for future use. |
| FLAGS_USER | UINT16 | Additional flags field defining user set hardware capabilities and features.<br>Bit values are… |
| • FLAG_CMD_CSUM_DISABLE | | When true, the command processor ignores checksums at the end of received commands.<br>This mode is useful for developers wanting to enter commands manually through a terminal application (where it is difficult to compute the CRC16 values on the fly). |
| • FLAG_MAG_SENS_DISABLE | | When true, this flag disables the function of the magnetic “reset-to-defaults” sensor. |

### 6.3.17. IPADDR_T (IP v4 Address)

The network IP address structure can be defined either an array of 4 sequential bytes or a single UINT32 value (or a union of both).

In C++, this definition could be represented as…

```
struct IpAddr_T {
	union {
		uint32 Addr;
		uint8 Bytes[4];
	};
};
```

| *Parameter* | *Type* | *Description* |
|---|---|---|
| BYTES | UINT8[4] | The 4 IP address fields in reverse order to the writing convention. (i.e. “<byte3> . <byte2> . <byte1> . <byte0>”) |
| *Or (Union)…* | | |
| ADDR | UINT32 | Little Endian representation of IP address fields stored in reverse order. |

For example an IP address of 192.168.0.1 would be stored in an array as…

IP[0] = 1, IP[1] = 0, IP[2] = 168, IP[3] = 192

In turn, this array would map to a little-endian UINT32 value as…

IPVAL = (192 << 24) + (168 << 16) + (0 << 8) + (1 << 0)

IPVAL = 3221225472 + 11010048 + 0 + 1

IPVAL = 3232235521

### 6.3.18. MACADDR_T (MAC Address)

Network MAC addresses normally only require 6-bytes of memory allocation. However, for convenience the SeaTrac beacon treats MAC addresses as a UINT64 type, but uses a union to allow overlaying with a 6-byte sequential array.

In C++, this definition could be represented as…

```
struct MacAddr_T {
	union {
		uint64 Addr;
		uint8 Bytes[6];
	};
};
```

| *Parameter* | *Type* | *Description* |
|---|---|---|
| BYTES | UINT8[6] | The MAC address fields in reverse order to the writing convention.<br>(i.e. “<byte5> - <byte4> - <byte3> - <byte2> - <byte1> - <byte0>”) |
| *Or (Union)…* | | |
| ADDR | UINT64 | Little Endian representation of the MAC address fields stored in reverse order. |

### 6.3.19. NAV_QUERY_T (NAV Protocol Query Bit Mask)

The NAV Protocol Query Flags type is defined as a bit-field stored in a UINT8 value, where one or more bits (flags) may be set to specify an overall numerical value.
Bits are defined as…

| | | |
|---|---|---|
| <value> | UINT8 | A bitmask specifying data to request or data included in a response.<br><br>Bit values are… |
| • Bits[7] = QRY_DATA | | When set, a NAV_QUERY_SEND command will request that any queued pending NAV data should be sent back, and a NAV_QUERY_RESP will contain data payload fields. |
| • Bit[6] = QRY_DATA_PLUS | | When set, a NAV_QUERY_SEND command will request that any queued NAV data should be sent back as well as the other requested fields (as far as will fit). When set in a NAV_QUERY_RESP, the response will contain a data payload, and possibly other fields as specified by the other flags. |
| • Bits[5:4] = RESERVED | | Reserved for future use, treat as 0's. |
| • Bit[3] = QRY_ATTITUDE | | When set, a NAV_QUERY_SEND command will request that attitude information is sent back, and a NAV_QUERY_RESP will contain attitude data fields. |
| • Bit[2] = QRY_TEMP | | When set, a NAV_QUERY_SEND command will request that temperature information is sent back, and a NAV_QUERY_RESP will contain temperature data fields. |
| • Bit[1] = QRY_SUPPLY | | When set, a NAV_QUERY_SEND command will request that supply voltage information is sent back, and a NAV_QUERY_RESP will contain supply voltage data fields. |
| • Bit[0] = QRY_DEPTH | | When set, a NAV_QUERY_SEND command will request that depth information is sent back, and a NAV_QUERY_RESP will contain depth data fields. |

### 6.3.20. PRESSURE_CAL_T (Pressure Sensor Calibration)

Holds calibration information for the Pressure Sensor. Fields are…

| *Parameter* | *Type* | *Description* |
|---|---|---|
| ID | UINT32 | The unique serial number of the sensor |

| | | |
|---|---|---|
| TYPE | UINT8 | The type of pressure sensor. Value should be 1 (PA type) |
| PRESSURE_MIN | INT16 | The minimum pressure reading in deciBars. |
| PRESSURE_MAX | INT16 | The maximum pressure reading in deciBars. |
| CAL_DAY | UINT8 | The day (1-32) of calibration |
| CAL_MONTH | UINT8 | The month (1-12) of calibration |
| CAL_YEAR | UINT16 | The year of calibration |

### 6.3.21. SETTINGS_T (Settings Record Structure)

The Settings Record structure is used to either retrieve the current working settings values in use from the beacon, or apply new changes to the beacon.

As this structure contains current calibration and communication settings for the beacon, it is always recommended to read the contents of the structure from the beacon rather than populating a new structure from scratch.
See Settings manipulation commands in section 7.4 from 99 for further details.



Settings marked with the "★" symbol are only applied on power-up or following a CID_SYS_REBOOT command.

| *Parameter* | *Type* | *Description* |
|---|---|---|
| STATUS_FLAGS | UINT8 | Value containing flags that control the generation and output of CID_STATUS messages.<br><br>Bit values are… |
| • Bits[7:3] = RESERVED | | Reserved for future use, treat as 0's. |
| • Bits[2:0] = STATUS_MODE | | Specifies how often periodic status output messages are generated. Mask and then treat as a STATUSMODE_E value. |
| STATUS_OUTPUT | STATUS_BITS_T | A bit-mask specifying which information should be included in generated status output messages.<br>For each bit set in this mask, a corresponding group of output fields will be appended to status messages (making them larger in size).<br>For details of how these fields affect the status message content, see CID_STATUS. |
| UART_MAIN_BAUD★ | BAUDRATE_E | Specifies the serial baud rate to be used by the main communications port. |
| UART_AUX_BAUD★ | BAUDRATE_E | Reserved for future use.<br>When populating this field, use a default value of BAUD_115200. |
| NET_MAC_ADDR★ | MACADDR_T | Reserved for future use.<br>When populating this structure, use a default value of 0. |
| NET_IP_ADDR★ | IPADDR_T | Reserved for future use.<br>When populating this structure, use a default value of 0xC0A801FA (192.168.1.250). |

| | | |
|---|---|---|
| NET_IP_SUBNET★ | IPADDR_T | Reserved for future use.<br>When populating this structure, use a value of 0xFFFF0000 (255.255.0.0). |
| NET_IP_GATEWAY★ | IPADDR_T | Reserved for future use.<br>When populating this structure, use a default value of 0xC0A80101 (192.168.1.1). |
| NET_IP_DNS★ | IPADDR_T | Reserved for future use.<br>When populating this structure, use a default value of 0xC0A80101 (192.168.1.1). |
| NET_TCP_PORT★ | UINT16 | Reserved for future use.<br>When populating this structure, use a default value of 8100. |
| ENV_FLAGS | UINT8 | This value contains flags that control the processing of the beacons environmental sensors (pressure, temperature, supply voltage etc).<br><br>Bit values are… |
| • Bits[7:2] = RESERVED | | Reserved for future use. |
| • Bit[1] = AUTO_PRESSURE_OFS | | When this flag is true, the pressure offset is automatically chosen using the minimum observed pressure reading when the beacon is in less than 3m of water (0.3 bar).<br>The assumption is than when fitted to an ROV this value will be seen on deck after the ROV is powered up, but if power is cycled when the beacon is below 3m, the pressure offset will not be updated. |
| • Bit[0] = AUTO_VOS | | When this flag is true, the velocity-of-sound used in range timing equations is automatically computed form the current water pressure, temperature and manually specified salinity.<br><br>VOS is calculated using the Coppens 1981 equation where temperature is valid over 0°C to 45°C, salinity over 0ptt to 35ppt and depth over 0m to 4000m. |

| | | |
|---|---|---|
| ENV_PRESSURE_OFS | INT32 | The manually specified offset applied to readings take from the pressure sensor to compensate for altitude and atmospheric pressure changes.<br><br>Values are encoded in milli-Bars, so divide by 1000 to obtain a value in Bars.<br>Valid values lie in the range -1 to 1000 Bar.<br><br>If auto-computation of pressure offset is enabled (in ENV_FLAGS), then any value written to this field will be lost the next time the offset is calculated. |
| ENV_SALINITY | UINT16 | The salinity value used when computing the velocity-of-sound from current pressure/depth.<br><br>Values are encoded as deci-parts-per-thousand (i.e. a value of 345 represents 34.5 ppt), so divide this value by 10 to obtain a value in ppt.<br><br>Typically a value of 0 represents fresh water, while 350 (35ppt) represents average sea water.<br>Values are valid in the range 0 to 100ppt.<br><br>If auto-computation of VOS is disabled (in ENV_FLAGS) then this value is not used. |
| ENV_VOS | UINT16 | The manually specified velocity of sound (VOS) to be used to convert range timing information into distances.<br><br>Values are encoded in deci-metres-per-second, so divide by 10 to obtain a value in metres-per-second.<br>Valid values are in the range 100ms$^{-1}$ to 2000ms$^{-1}$.<br><br>If auto-computation of VOS is enabled (in ENV_FLAGS), then any value written to this field will be lost the next time the VOS is calculated. |
| AHRS_FLAGS | UINT8 | This value contains flags that control the operation of the beacons AHRS system.<br><br>Bit values are… |
| • Bits[7:1] = RESERVED | | Reserved for future use. |

| | | |
|---|---|---|
| • Bit[0] = AUTO_CAL_MAG | | When this bit is true, automatic (dynamic) calibration of the magnetometer is enabled.<br><br>In this mode, the magnetic field surrounding the beacon is continuously samples as the beacon is rotated through space, and every 30s a new calibration is attempted. If the results are better than the current calibration, then the new coefficients are accepted. |
| AHRS_CAL | AHRSCAL_T | Structure containing the calibration data for the Attitude/Heading Reference System (AHRS). |
| AHRS_YAW_OFS | UINT16 | A fixed attitude yaw offset that is applied to all AHRS reading. Offsets are applied to the AHRS output via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are cyclically wrapped to the range 0° to 359.9°. |
| AHRS_PITCH_OFS | UINT16 | A fixed attitude pitch offset that is applied to all AHRS reading. Offsets are applied to the AHRS output via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are clipped to the range -90.0° to +90.0°. |
| AHRS_ROLL_OFS | UINT16 | A fixed attitude roll offset that is applied to all AHRS reading. Offsets are applied to the AHRS output via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are clipped to the range -180.0° to +180.0°. |
| XCVR_FLAGS | UINT8 | Value containing flags to control the operation of the acoustic transceiver.<br><br>Bit values are… |

| | |
|---|---|
| • Bit[7] = XCVR_DIAG_MSGS | When this flag is true a series of diagnostic status messages will be generated by triggering events processed by the acoustic transceiver – for further details see the following commands…<br>CID_XCVR_TX_MSG, CID_XCVR_RX_ERR<br>CID_XCVR_RX_MSG, CID_XCVR_RX_REQ<br>CID_XCVR_RX_RESP, CID_XCVR_RX_RESP_ERROR<br>CID_XCVR_RX_UNHANDLED |
| • Bit[6] = XCVR_FIX_MSGS | When this flag is true, a CID_XCVR_FIX status message will be generated on successful reception of an acoustic response message.<br>The fix message contains details relating to distance, position and depth of the remote beacon. |
| • Bit[5] = XCVR_USBL_MSGS | When this flag is true, a CID_XCVR_USBL status message is generated on successfully reception of an acoustic message containing USBL signal information. |
| • Bits[4:3] = XCVR_TX_MSGCTRL | Transmit Message control flags. These allow the system to control which message transmissions are allowed or prohibited.<br>For bit value details see the XCVR_TXMSGCTRL_E enumeration. |
| • Bit[2] = XCVR_BASELINES_MSGS | When this flag is true, a CID_XCVR_BASELINES status message is generated on successful reception of an acoustic message. The baselines message contains the required information to calculate a position.<br>When this message is enabled, the beacon will not perform its own internal position calculation, the SDK must handle this. |
| • Bit[1] = XCVR_POSFLT_ENABLE | When this flag is true, the position filter is enabled to mark potentially erroneous acoustic USBL fixes based on velocity and angular movement limits. |
| • Bit[0] = USBL_USE_AHRS | When this flag is true the acoustic transceiver will use the current AHRS attitude (updated internally at a 50Hz rate) when resolving relative positions of remote beacons to the local beacon.<br><br>When the flag is false, the fixed attitude specified in the XCVR_YAW, XCVR_PITCH and XCVR_ROLL fields will be used. |

| | | |
|---|---|---|
| XCVR_BEACON_ID | BID_E | The identification code the local beacon will accept messages addressed to, or use as the source identifier when sending messages.<br><br>Valid values are from 1 to 100 (0x1 to 0xF). A value of 0 (BEACON_ALL) should not be used. |
| XCVR_RANGE_TMO | UINT16 | The range timeout specifies a distance (related to time by the VOS setting) beyond which responses from beacons are ignored, and the remote beacon is considered to have timed out (see CID_XCVR_RX_RESP_ERROR messages).<br><br>Values are encoded in metres.<br>Valid values are in the range 100m to 3000m. |
| XCVR_RESP_TIME | UINT16 | The response turnaround time specifies how long the beacon will wait between receiving a request message and starting transmission of the response message.<br>All beacons communicating acoustically within the same network must use the same value otherwise range errors will be observed.<br><br>Typically, larger values than the default of 10ms can be used to reduce multi-path issues in confined spaces and allow echoes to die down before the response is sent, but should only be adjust if communication reliability issues are observed.<br><br>Values are encoded in milliseconds.<br>Valid values are in the range 10ms to 1000ms. |
| XCVR_YAW | UINT16 | When the AHRS attitude is not used to specify the transceiver attitude, this value is used as the manually specified yaw attitude from which relative positions of remote beacons to the local beacon are computed.<br>Attitudes are applied in the position calculation routine via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are cyclically wrapped to the range 0° to 359.9°. |

| | | |
|---|---|---|
| XCVR_PITCH | UINT16 | When the AHRS attitude is not used to specify the transceiver attitude, this value is used as the manually specified pitch attitude from which relative positions of remote beacons to the local beacon are computed.<br>Attitudes are applied in the position calculation routine via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are clipped to the range -90.0° to +90.0°. |
| XCVR_ROLL | UINT16 | When the AHRS attitude is not used to specify the transceiver attitude, this value is used as the manually specified roll attitude from which relative positions of remote beacons to the local beacon are computed.<br>Attitudes are applied in the position calculation routine via a Direction-Cosine-Matrix, in the Euler sequence Yaw, Pitch then Roll.<br><br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees.<br>Valid values are clipped to the range -180.0° to +180.0°. |
| XCVR_POSFLT_VEL | UINT8 | The maximum velocity limit (in metres per second) that the position filter expects to see a beacon move at. Position Fix outputs for Beacons that have moved faster than this in the time between pings will be marked as a position error. |
| XCVR_POSFLT_ANG | UINT8 | For beacons that are further away, azimuth errors start to come into play. This value defines the angular limits that beacons can move (or position jitter) within without being marked as an error.<br>Vales are specified in degrees, and typically this value is 10 degrees. |

| | | |
|---|---|---|
| XCVR_POSFLT_TMO | UINT8 | This timeout limit specified in seconds that maximum time that a beacon is not communicated with before its position filter is reset, allowing its next position (what ever that may be) to be marked as valid.<br>For example, a value of 60 seconds means that if no communications have been made with the beacon for 60 seconds, then its position could be far outside the limits expected by the position filter, so allow its position and restart tracking on the next fix. |

### 6.3.22. STATUS_BITS_T (Status Fields Bit-Mask)

The Status Flags type is defined as a bit-field stored in a UINT8 value, where one or more bits (flags) may be set to specify an overall numerical value.
Bits are defined as…

| | | |
|---|---|---|
| <value> | UINT8 | A bit-mask specifying which information should be included in generated status output messages.<br><br>Bit values are… |
| • Bits[7:6] = RESERVED | | Reserved for future use, treat as 0's. |
| • Bit[5] = AHRS_COMP_DATA | | When set, appends compensated sensor data fields to the end of the status output message. |
| • Bit[4] = AHRS_RAW_DATA | | When set, appends raw sensor data fields to the end of the status output message. |
| • Bit[3] = ACC_CAL | | When set, appends accelerometer calibration and limits fields to the end of the status output message. |
| • Bit[2] = MAG_CAL | | When set, appends magnetometer calibration and buffer fields to the end of the status output message. |
| • Bit[1] = ATTITUDE | | When set, appends the AHRS attitude (yaw, pitch, roll) fields to the end of the status output message. |
| • Bit[0] = ENVIRONMENT | | When set, appends environmental sensor data fields (temperature, depth, VOS, supply voltage etc) to the end of the status output message. |

# 7. Beacon Management Message Definitions

## 7.1. System Messages

System messages are the core set of command supported by the beacon in both its normal operating mode and in Bootloader mode.
These commands provide basic functions to determine if the device is response, identify its hardware/firmware configuration, and allow firmware reprogramming.

### 7.1.1. CID_SYS_ALIVE

The CID_SYS_ALIVE command is a simple command that can be periodically sent to the beacon to determine if the hardware is present, connected and functioning.
For example, a host system sending the CID_SYS_ALIVE command at regular intervals can use the response reply to reset a “connection lost” timeout timer, so that when a connection fails the lack of responses can indicate to the user the hardware is no longer present.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_ALIVE) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_ALIVE) |
| SECONDS | UINT32 | The number of seconds the beacon has been powered up for. |

### 7.1.2. CID_SYS_INFO

The CID_SYS_INFO command is used to receive hardware and firmware identification information from the beacon. This information can then be used by external applications to determine if firmware needs updating, or to determine support for Command Messages based on the hardware type.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_INFO) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_INFO) |
| SECONDS | UINT32 | The number of seconds the beacon has been powered up for. |
| SECTION | UINT8 | The memory section that the current firmware is executing in. Valid values are…<br>• 0 = Bootloader Application<br>• 1 = Main Application |
| HARDWARE | HARDWARE_T | Structure containing details about the Beacon hardware, including part number, revision and unique serial number. |
| BOOT_FIRMWARE | FIRMWARE_T | Structure containing details about the Bootloader firmware, including version, build and checksum details. |
| MAIN_FIRMWARE | FIRMWARE_T | Structure containing details about the Main Application firmware, including version, build and checksum details. |
| BOARD_REV | UINT8 | Indicates the type of circuit board fitted in the beacon.<br>• 0x00 - X150 & X110 rev 1 & 2 boards, X010 rev 1 board.<br>• 0x01 - X150 & X110 rev 3 boards, X010 rev 2 board. |

Depending on the application executing, additional information follows…

If running in SECTION contains 0 (Bootloader application mode), the following will be output…

| | | |
|---|---|---|
| EXTENDED_INFO | UINT8 | Value will be 0x00 indicating no further information |

If running in SECTION contains 1 (Main application mode), the following will be output…

| | | |
|---|---|---|
| EXTENDED_INFO | UINT8 | Value will be 0xFF indicating additional information |
| FLAGS | UINT8 | Reads as 0x01 (for Pressure Sensor information present) |
| RESERVED | UINT8[3] | 3 bytes reserved for future use – read as 0's |
| PRESSURE_SENSOR | PRESSURE_CAL_T | Structure contains information on the pressure sensor (serial number, max operating depth etc). |

### 7.1.3. CID_SYS_REBOOT

The CID_SYS_REBOOT command is sent to perform a software reset of the beacon. To prevent accidental resets, the command requires an additional UINT16 constant value to be specified after the CID code.

On receiving a valid reset command, the beacon performs the following actions…

- The status LED turns yellow
- The response message is transmitted
- The serial port is closed, so no further commands can be decoded.
- A 500ms shut-down period is observed.
- The beacon restarts, any communications settings changes made will be applied.
- Human readable power-up diagnostic information is output from the serial port.
- The beacon is ready to receive further commands.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_REBOOT) |
| CHECK | UINT16 | Check constant value used to confirm a reset should be performed. This value should be 0x6A95. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SYS_REBOOT) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_CMD_PARAM_MISSING - The CHECK parameter was not specified in the Command Message.<br>• CST_CMD_PARAM_INVALID - The CHECK parameter was incorrect in the Command Message. |

### 7.1.4. CID_SYS_ENGINEERING

The CID_SYS_ENGINEERING command is used for factory commissioning, diagnostic and debugging purposes that lie outside the scope of this document.

## 7.2. Firmware Programming Messages

For developers wishing to allow beacons firmware to be updated while in the field and connected to their system/software, the following 3 commands provide the facility to transfer a SeaTrac XML Firmware file through the serial link.

Opening the FWX firmware file in a text editor will show and XML structure and content similar to below…

```
<?xml version="1.0" encoding="UTF-8"?>
<Firmware>
	<Config>
		<Timestamp>2014-09-01 14:02:56</Timestamp>
	</Config>
	<Targets>
		<Target
			Device="0"
			Name="SeaTrac X-Series Application"
			Timestamp="2014-09-01 13:45:16"
			Section="1"
			DataFormat="2"
			Length="157552"
			Checksum="4DBF60E9"
			Signature="685CBFB440C6F3F8EF05CA43B74E3134D7C92C23"
		>
			<Data Block="0" Length="64">
				01DB3154998D1DFF0123BC00......67F9FFAE
			</Data>
			<Data Block="1" Length="64">
				A91BAF09C059040E6083238A......5B9CED86
			</Data>
			<Data Block="2" Length="64">
				1368C707E27640C79670B864......F6578B0D
			</Data>


			<Data Block="2459" Length="64">
				98E9928240C339AD897775C1......493A788F
			</Data>
			<Data Block="2460" Length="64">
				C7D736BF5AEEF6AB0C84CBC0......54045579
			</Data>
			<Data Block="2461" Length="48">
				0E81E5ED133F......9E7B53F8
			</Data>
		</Target>
	</Targets>
</Firmware>
```

The file should be processed using an XML parser which will yield the following nodes…

- FIMWARE — This is the top level document node containing the CONFIG and TARGET node.
- CONFIG — This node contains overall configuration information for the firmware file. At present the only TIMESTAMP sub node contains the build time-stamp of the firmware.
- TARGETS — The TARGETS node contains a list of firmware applicable to each physical device that can be programmed on the hardware.
  Each device has a corresponding TARGET node definition.
- TARGET — The TARGET node contains firmware for each device. The attributes of the target node define the overall properties of the firmware used by the programming utility, and several of these are required to be sent when starting programming using the CID_PROG_INIT command.
- DATA — The DATA nodes for the firmware target contains pre-formatted ASCII-HEX data strings that should be sent to the device being programmed using CID_PROG_BLOCK commands.

### 7.2.1. CID_PROG_INIT

The CID_PROG_INIT command is used to initialise the firmware programming sequence. Calling this command will instruct the beacon to shut down the AHRS and Acoustic Transceiver systems until programming is complete.

That command is sent along with several parameters that are obtained by parsing TARGET XML node from the firmware FWX file provided by Blueprint. The beacon will examine these parameters and determine if the firmware file is applicable to it and if it is ready to accept the update.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_INIT) |
| FW_SECTION | UINT8 | Value obtained from the “Section” attribute of the FWX file TARGET node. |
| FW_PART_NUMBER | UINT16 | Value obtained from the “PartNumber” attribute of the FWX file TARGET node. If the attribute is not specified in the file, use a value of 0. |
| FW_PART_REV_MIN | UINT8 | Value obtained from the “PartRevMin” attribute of the FWX file TARGET node. If the attribute is not specified in the file, use a value of 0. |
| FW_PART_REV_MAX | UINT8 | Value obtained from the “PartRevMax” attribute of the FWX file TARGET node. If the attribute is not specified in the file, use a value of 0. |

| | | |
|---|---|---|
| FW_SERIAL_NUMBER | UINT32 | Value obtained from the “SerialNumber” attribute of the FWX file TARGET node. If the attribute is not specified in the file, use a value of 0. |
| FW_DATA_FORMAT | UINT8 | Value obtained from the “DataFormat” attribute of the FWX file TARGET node. |
| FW_LENGTH | UINT32 | Value obtained from the “Length” attribute of the FWX file TARGET node. |
| FW_CHECKSUM | UINT32 | Value obtained from the “Checksum” attribute of the FWX file TARGET node. |
| FW_SIGNATURE | UINT8[20] | Value obtained from the “Signature” attribute of the FWX file TARGET node.<br>The signature attribute contains 40 ASCII-Hex characters describing 20 bytes of data, so the string should be first decoded into an array of bytes that are then stored sequentially in this field. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_INIT) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully, and the device is ready to accept further CID_PROG_BLOCK commands.<br>• CST_CMD_PARAM_MISSING – One or more of the command parameters is formatted incorrectly.<br>• CST_PROG_FLASH_ERROR – A problem with flash memory is preventing programming.<br>• CST_PROG_FIRMWARE_ERROR – There is a problem with the firmware credentials supplied, and the beacon will not accept the new firmware.<br>• CST_PROG_SECTION_ERROR – The section parameter is invalid, or you are trying to program the Bootloader section while executing code from the Bootloader section. |

- CST_PROG_LENGTH_ERROR – The length parameter is too large to fit into the available memory
- CST_FAIL – An unspecified firmware error has occurred.

### 7.2.2. CID_PROG_BLOCK

Once the CID_PROG_INIT command has been sent and accepted successfully by the beacon (returning a STATUS code of CST_OK), the CID_PROG_BLOCK command is then sent repeatedly to transfer each DATA node found in the FWX file under the required TARGET node.

Data in the FWX file is encoded in ASCII-HEX pairs, so for a DATA node length attribute of 64, there will 128 ASCII-HEX characters.

A CID_PROG_BLOCK command should be sent for each DATA node in sequence, and a valid STATUS code should be received back from the beacon before sending the next block.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_BLOCK) |
| DATA_LENGTH | UINT16 | The number of bytes that will follow in this message. |
| DATA | UINT8[x] | An array of data bytes read from the relevant DATA node in the FWX file, where “x” is the number of bytes specified by DATA_LENGTH. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_BLOCK) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully and the next CID_PROG_BLOCK command can be sent.<br>• CST_CMD_PARAM_MISSING – One or more of the command parameters is formatted incorrectly. |

- CST_CMD_PARAM_INVALID – The length of data specified if too large.
- CST_PROG_FLASH_ERROR – A problem with flash memory is preventing programming.
- CST_PROG_INIT_ERROR – There beacon has not initialised programming, send a CID_PROG_INIT command and get a valid CST_OK response.
- CST_PROG_LENGTH_ERROR – The amount of data sent has exceeded the amount of firmware storage available.
- CST_FAIL – An unspecified firmware error has occurred.

### 7.2.3. CID_PROG_UPDATE

Once all the CID_PROG_BLOCK commands have been sent in the correct sequence, the CID_PROG_UPDATE command is used to transfer the new firmware from the working download area into active memory.

This command may take up to 5 seconds to execute depending on the size of the firmware being programmed.

If the command executes successfully, the status LED will illuminate yellow during the update process, then the beacon will reboot.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_UPDATE) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PROG_UPDATE) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully and the firmware has been updated.<br>• CST_PROG_FLASH_ERROR – A problem with flash memory is preventing programming.<br>• CST_PROG_INIT_ERROR – There beacon has not initialised programming, send a CID_PROG_INIT command and get a valid CST_OK response.<br>• CST_PROG_LENGTH_ERROR – The length parameter is too large to fit into the available memory<br>• CST_PROG_CHECKSUM_ERROR – The Checksum supplied with CID_PROG_INIT does not match the checksum of data downloaded with the CID_PROG_BLOCK commands.<br>• CST_FAIL – An unspecified firmware error has occurred. |

## 7.3. Status Messages

The Status Messaging system allows external applications to display or log the real-time sensor readings and operational parameters of the beacon.

Status Messages can be either manually requested by issuing the CID_STATUS command, or set to be automatically generated at specified time intervals (or disabled) by settings configured with the CID_SETTINGS_SET or CID_STATUS_CFG_SET commands.

The allow optimisation of communication link bandwidth or reduce data overhead for logging purposes, the information content of status messages it split into distinct groups (see STATUS_BITS_T) and can be specified either in the system settings or at the time a manual status request is made.

### 7.3.1. CID_STATUS

The CID_STATUS message serves several purposes: when issued as a command, it requests the current beacon status at that point in time, or alternately it can be set to be output periodically at specified intervals (through the STATUS_OUTPUT field of either the CID_SETTINGS_SET or CID_STATUS_CFG_SET commands).

When issued as a manual status request command, an optional STATUS_OUTPUT value can be specified in the CID_STATUS message to determine the information content of the response message.
Omitting the STATUS_OUTPUT field causes the beacon to use the currently configured settings option specified with the CID_SETTINGS_SET or CID_STATUS_CFG_SET commands. Alternately, if the STATUS_OUTPUT value is specified, this will not update the stored settings value or alter if status messages are also periodically generated.



This message is a status message that may be sent by the beacon at periodic time intervals (not in response to a command message) depending on STATUS_OUTPUT settings field – see SETTINGS_T and the CID_SETTINGS_SET or CID_STATUS_CFG_SET commands.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS) |
| STATUS_OUTPUT (optional) | STATUS_BITS_T | An optional bit-mask specifying which information should be included in the generated status output response message. For each bit set in this mask, a corresponding group of output fields will be appended to status message (making it larger in size). |

**Response/Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS) |
| STATUS_OUTPUT | STATUS_BITS_T | A bit-mask specifying which information is included in the following status output message, so any decoding algorithm can correctly parse the message record. |
| TIMESTAMP | UINT64 | The value of the beacon's clock (set to zero when the beacon was powered up – not rebooted!).<br>Values are encoded in milliseconds, so divide by 1000 for seconds. |

### Environmental Fields

If the message STATUS_OUTPUT parameter contains the ENVIRONMENT bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record…

| | | |
|---|---|---|
| ENV_SUPPLY | UINT16 | The beacons supply voltage.<br>Values are encoded in milli-volts, so divide by 1000 for a value in Volts. |
| ENV_TEMP | INT16 | The temperature of air/water in contact with the diaphragm of the pressure sensor.<br>Values are encoded in deci-Celsius, so divide by 10 to obtain a value in Celsius. |
| ENV_PRESSURE | INT32 | The external pressure measured on the diaphragm of the pressure sensor.<br>Values are encoded in milli-bars, so divide by 1000 to obtain a value in Bar.<br><br>Please note, the specification of pressure reading is 0.5% of the sensors full-scale value, so for a 200 Bar sensor the tolerance applicable to this value is ±1 Bar (~10m). |
| ENV_DEPTH | INT32 | The computed depth based on the measured environmental pressure.<br>Values are encoded in deci-metres, so divide by 10 for a value in metres. |
| ENV_VOS | UINT16 | The value of Velocity-of-Sound currently being used for computing ranges.<br>This may be either the manually specified VOS from settings, or the auto-computed value based on pressure, temperature and salinity. |

| | | |
|---|---|---|
| | | Values are encoded in deci-metres-per-second, so divide by 10 to obtain a value in metres-per-second. |

### Attitude Fields

If the message STATUS_OUTPUT parameter contains the ATTITUDE bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record.
For details of attitude definitions refer to section 9.1 on page 184.

| | | |
|---|---|---|
| ATT_YAW | INT16 | The current Yaw angle of the beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |
| ATT_PITCH | INT16 | The current Pitch angle of the beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |
| ATT_ROLL | INT16 | The current Roll angle of the beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |

### Magnetometer Calibration and Status Fields

If the message STATUS_OUTPUT parameter contains the MAG_CAL bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record.
These fields are commonly used to monitor the current magnetic calibration state and to assist with the magnetometer calibration procedure.

| | | |
|---|---|---|
| MAG_CAL_BUF | UINT8 | Value that indicates how full the data buffer is that holds magnetometer values describing the surrounding magnetic environment.<br>Values are encoded as a percentage from 0 to 100 representing empty (where no magnetic calibration is possible) to full (where the best magnetic calibration can be computed). |
| MAG_CAL_VALID | BOOLEAN | The flag is True if a magnetic calibration has been computed and is currently in use, compensating magnetometer readings. |
| MAG_CAL_AGE | UINT32 | The number of seconds that have elapsed since the magnetometer calibration was last computed. |

| | | |
|---|---|---|
| | | When dynamic calibration is enabled, and there is sufficient data in the magnetic calibration buffer, then calibrations should be computed every 30 seconds. |
| MAG_CAL_FIT | UINT8 | Value indicating how well the current magnetometer calibration can fit the measured data to an ideal “sphere” (or perfect calibration).<br>Values are encoded as a percentage from 0 to 100. |

### Accelerometer Calibration Fields

If the message STATUS_OUTPUT parameter contains the ACC_CAL bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record.
The fields are commonly used to assist in calibrating the accelerometer hardware.
For details of axis definitions refer to section 9.3 on page 186.

| | | |
|---|---|---|
| ACC_LIM_MIN_X | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent -1G on the X sensor axis. |
| ACC_LIM_MIN_Y | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent +1G on the X sensor axis. |
| ACC_LIM_MIN_Z | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent -1G on the Y sensor axis. |
| ACC_LIM_MAX_X | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent +1G on the Y sensor axis. |
| ACC_LIM_MAX_Y | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent -1G on the Z sensor axis. |
| ACC_LIM_MAX_Z | INT16 | Value that holds the raw accelerometer sensor value that will be used to represent +1G on the Z sensor axis. |

### Raw AHRS Sensor Data Fields

If the message STATUS_OUTPUT parameter contains the AHRS_RAW_DATA bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record.
For details of axis definitions refer to section 9.3 on page 186.
Values are sampled internally by the AHRS at a rate of 50Hz.

| | | |
|---|---|---|
| AHRS_RAW_ACC_X | INT16 | The last raw accelerometer sensor value measured on the X-axis.<br>This field is used during functional testing and can be used to assist with the accelerometer calibration procedure. |

| | | |
|---|---|---|
| | | Computing a ratio between this value and the -1G to +1G interval (specified by the ACC_LIM_MIN_X and ACC_LIM_MAX_X values), gives the current gravitation acceleration seen on the sensor axis. |
| AHRS_RAW_ACC_Y | INT16 | The last raw accelerometer sensor value measured on the Y-axis.<br>This field is used during functional testing and can be used to assist with the accelerometer calibration procedure.<br><br>Computing a ratio between this value and the -1G to +1G interval (specified by the ACC_LIM_MIN_Y and ACC_LIM_MAX_Y values), gives the current gravitation acceleration seen on the sensor axis. |
| AHRS_RAW_ACC_Z | INT16 | The last raw accelerometer sensor value measured on the Z-axis.<br>This field is used during functional testing and can be used to assist with the accelerometer calibration procedure.<br><br>Computing a ratio between this value and the -1G to +1G interval (specified by the ACC_LIM_MIN_Z and ACC_LIM_MAX_Z values), gives the current gravitation acceleration seen on the sensor axis. |
| AHRS_RAW_MAG_X | INT16 | The last raw magnetometer sensor value measure on the X-axis.<br>This field is used during functional testing and can be used to assist with the magnetometer calibration procedure (in conjunction with the accelerometer orientation value). |
| AHRS_RAW_MAG_Y | INT16 | The last raw magnetometer sensor value measure on the Y-axis.<br>This field is used during functional testing and can be used to assist with the magnetometer calibration procedure (in conjunction with the accelerometer orientation value). |
| AHRS_RAW_MAG_Z | INT16 | The last raw magnetometer sensor value measure on the Z-axis.<br>This field is used during functional testing and can be used to assist with the magnetometer calibration procedure (in conjunction with the accelerometer orientation value). |
| AHRS_RAW_GYRO_X | INT16 | The last raw rate of rotation measured around the X-axis of the gyroscope sensor. |

| | | |
|---|---|---|
| | | Values are encoded in degrees-per-second. |
| AHRS_RAW_GYRO_Y | INT16 | The last raw rate of rotation measured around the Y-axis of the gyroscope sensor.<br>Values are encoded in degrees-per-second. |
| AHRS_RAW_GYRO_Z | INT16 | The last raw rate of rotation measured around the Z-axis of the gyroscope sensor.<br>Values are encoded in degrees-per-second. |

### Compensated AHRS Sensor Data Fields

If the message STATUS_OUTPUT parameter contains the AHRS_COMP_DATA bit (see STATUS_BITS_T), then the following fields are sequentially appended to the message record.
For details of axis definitions refer to section 9.3 on page 186.
Values are sampled internally by the AHRS at a rate of 50Hz.

| | | |
|---|---|---|
| AHRS_COMP_ACC_X | FLOAT | The AHRS_RAW_ACC_X sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_ACC_Y | FLOAT | The AHRS_RAW_ACC_Y sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_ACC_Z | FLOAT | The AHRS_RAW_ACC_Z sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_MAG_X | FLOAT | The AHRS_RAW_MAG_X sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_MAG_Y | FLOAT | The AHRS_RAW_MAG_Y sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_MAG_Z | FLOAT | The AHRS_RAW_MAG_Z sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_GYRO_X | FLOAT | The AHRS_RAW_GYRO_X sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_GYRO_Y | FLOAT | The AHRS_RAW_ GYRO _Y sensor reading after the calibration coefficients have been applied. |
| AHRS_COMP_GYRO_Z | FLOAT | The AHRS_RAW_ GYRO _Z sensor reading after the calibration coefficients have been applied. |

### 7.3.2. CID_STATUS_CFG_GET

This CID_STATUS_CFG_GET command allows the current configuration for the generation of automated status messages to be quickly retrieved.

As an alternative to this command, use CID_SETTINGS_GET.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS_CFG_GET) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS_CFG_GET) |
| STATUS_OUTPUT | STATUS_BITS_T | A bit-mask specifying which information should be included in generated status output messages – see CID_STATUS. |
| STATUS_MODE | STATUSMODE_E | Specifies how often periodic status output messages are generated. |

### 7.3.3. CID_STATUS_CFG_SET

The CID_STATUS_CFG_SET command allows quick configuration of the status message output configuration without the need to read or reload the entire settings structure. However, any changes made with this command will only apply to the working RAM settings, and will be lost upon power-down unless they are subsequently with the CID_SETTINGS_SAVE command.

As an alternative to this command, use CID_SETTINGS_SET.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS_CFG_SET) |
| STATUS_OUTPUT | STATUS_BITS_T | A bit-mask specifying which information should be included in generated status output messages – see CID_STATUS. |
| STATUS_MODE | STATUSMODE_E | Specifies how often periodic status output messages are generated. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_STATUS_CFG_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_CMD_PARAM_MISSING – One of the parameters was not specified correctly. |

## 7.4. Settings Messages

The beacon settings system is responsible for the retrieval, verification, application and storage of parameters determining the operation of the beacon.

All settings are stored within a single “settings record” and at power-up this is loaded from permanent (non-volatile) EEPROM memory into the working RAM area.

Any changes made to the settings record by the user (through the CID_SETTINGS_SET command) are always made to the working RAM set and will be lost if the device is powered down before a CID_SETTINGS_SAVE command is issued to update the EEPROM copy.

When new settings are specified by the CID_SETTINGS_SET command most will be applied immediately. However, some settings such as those controlling communications baud-rates etc are only applied when the device is powered up (or restarts via a CID_SYS_REBOOT command). See the SETTINGS_T structure definition for details of which settings are only applied on power-up/reboot.



The EEPROM memory used to permanently store the settings information in the beacon has a lifetime endurance of between 10,000 and 50,000 write cycles, after which its data retention capabilities may start to degrade (below the guaranteed 5 year period).

For this reason, only store the working RAM settings when required with the CID_SETTINGS_SAVE command. When designing a control system, ensure that settings are only stored infrequently and when needed (i.e. not on a periodic basis that may reduce operating life).

### 7.4.1. CID_SETTINGS_GET

The CID_SETTINGS_GET command is issued to read back the settings record from working RAM containing the values currently in use.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_GET) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_GET) |
| SETTINGS | SETTINGS_T | Structure containing the current working RAM settings in use. |

### 7.4.2. CID_SETTINGS_SET

The CID_SETTINGS_SET command is issued to update the values of the working RAM settings and apply the new values.
This command updates the entire contents of the settings record, so for modification of settings it is recommended to first use CID_SETTINGS_GET command to read the current settings record, then modify and resend this.



This command does not save the new settings into permanent EEPROM storage and so changes made will be lost if the device is powered down or rebooted.
To store changes after the set command, use the CID_SETTINGS_SAVE command.



Not all settings will be applied after the set command is issued. Specifically communication baud-rate settings will only be applied when the device is next power up, or a CID_SYS_REBOOT command is issued.

See the SETTINGS_T structure definition for details of which settings are only applied on power-up/reboot.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_SET) |
| SETTINGS | SETTINGS_T | Structure containing the new settings to apply. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_CMD_PARAM_MISSING - The settings parameter were not specified correctly. |

### 7.4.3. CID_SETTINGS_LOAD

This CID_SETTINGS_LOAD command causes the beacon to re-load the working RAM settings from the EEPROM storage and apply them. Any previous changes made to the working RAM settings that have not been stored with the CID_SETTINGS_SAVE will be lost.



Not all settings will be applied after the load operation completes. Specifically communication baud-rate settings will only be applied when the device is next power up, or a CID_SYS_REBOOT command is issued.

See the SETTINGS_T structure definition for details of which settings are only applied on power-up/reboot.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_LOAD) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_LOAD) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_FAIL - The settings could not be loaded (possibly due to memory or data corruption), default settings have been used instead. |

### 7.4.4. CID_SETTINGS_SAVE

The CID_SETTINGS_SAVE command is sent to save the current working RAM settings into permanent storage, ensuring they are used next time the beacon is powered up.



The beacon uses EEPROM memory to permanently store the settings information, but this has a lifetime endurance of between 10,000 and 50,000 cycles, after which its data retention capabilities may start to degrade (below the guaranteed 5 year period).

For this reason, the beacon settings system allows quick and real-time parameter changes to be made into working RAM, and only stored when required with the CID_SETTINGS_SAVE command.
When designing a control system, ensure that settings are only stored infrequently and when needed (i.e. not on a periodic basis that may reduce operating life).



If the communications baud-rate settings are accidentally changed, saved and the device rebooted, it may not be possible to re-establish serial communications (without trying all available baud rates).
An alternative is to use the magnetic reset-to-defaults function supported by the beacon (although this will also reset calibration data) – see the beacon user manual for further details.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_SAVE) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_SAVE) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_FAIL - The settings could not be saved successfully (a storage error may have occurred). |

### 7.4.5. CID_SETTINGS_RESET

The CID_SETTINGS_RESET command is used to reset the working RAM settings back to their factory default values, and store the default values back into EEPROM memory.



The beacon uses EEPROM memory to permanently store the settings information, but this has a lifetime endurance of between 10,000 and 50,000 cycles, after which its data retention capabilities may start to degrade (below the guaranteed 5 year period).

When designing a control system, ensure that settings are only reset infrequently and when needed (i.e. not on a periodic basis that may reduce operating life).



Not all settings will be applied after the load operation completes. Specifically communication baud-rate settings will only be applied when the device is next power up, or a CID_SYS_REBOOT command is issued.

See the SETTINGS_T structure definition for details of which settings are only applied on power-up/reboot.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_RESET) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_SETTINGS_RESET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_FAIL - The settings have been reset, in RAM but could not be saved (possibly due to memory error). |

## 7.5. Calibration Messages

Occasionally the beacon will need its onboard sensor calibrating, specifically the magnetometer (to provider the magnetic north yaw heading for the AHRS system), and the pressure-sensor offset for the depth computation (to compensate for atmospheric pressure changes).

The following calibration commands have been provided to allow application and developers easy access to sensor calibration functions.

### 7.5.1. CID_CAL_ACTION

The Calibration Action command should be used at various times by external calibration applications and algorithms to instruct to beacon to perform operations related to one of its on-board sensors.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_SET) |
| ACTION | CAL_ACTION_E | The code of the calibration action that should be performed. See below for further details of each operation code. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_CMD_PARAM_MISSING – The ACTION parameter was not specified correctly. |

Summary of Calibration Actions…

- CAL_ACC_DEFAULTS — This operation sets the current accelerometer calibration coefficients back to their values (of 0) in the AHRS working AHRSCAL_T settings. However, this does not reset the currently measured sensor limit values (output in the CID_STATUS message), that are used to determine the calibration coefficients by the CAL_ACC_CALC action.

- CAL_ACC_RESET — This operation resets the measured accelerometer MIN and MAX filtered values measured by the sensor as it is slowly rotated during a calibration procedure (and output in the CID_STATUS message).
  However, this does not reset any of the accelerometer calibration coefficients in the working AHRSCAL_T settings.

- CAL_ACC_CALC — This action should be called once new sensor MIN and MAX limit values have been obtained by rotation during a accelerometer calibration.
  When issued, the beacon calculates the new calibration coefficients from the measured limits (CID_SETTINGS_SAVE should then be issued to store the new calibration to EEPROM memory).

- CAL_MAG_DEFAULTS — This action sets the current magnetometer calibration values for Hard and Sort Iron back to their default values, but does not clear the Magnetic Calibration Buffer.

- CAL_MAG_RESET — This action clears the magnetic calibration buffer, that is continuously measuring the surrounding magnetic environment as the beacon is rotated. However, this action does modify any of the current calibration coefficients.

- CAL_MAG_CALC — Once the magnetic buffer has been filled, calling this action will calculate and apply a new magnetometer calibration for Hard and Soft Iron compensation (CID_SETTINGS_SAVE should then be issued to store the new calibration to EEPROM memory).

- CAL_PRES_OFFSET_RESET — Performing this action resets the pressure-offset back to zero Bar.

- CAL_PRES_OFFSET_CALC — This action sets the pressure-offset value from the current pressure sensor reading – in effect zeroing the depth sensor.

### 7.5.2. CID_AHRS_CAL_GET

The CID_AHRS_CAL_GET command allows the current calibration coefficients in use by the AHRS system to be quickly retrieved.

As an alternative to this command, use CID_SETTINGS_GET.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_GET) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_GET) |
| AHRS_CAL | AHRSCAL_T | The structure containing the current calibration coefficients in use on the AHRS system. |

### 7.5.3. CID_AHRS_CAL_SET

The CID_AHRS_CAL_SET command is used to allow direct updating of the AHRS sensor calibration coefficients without the need to read or reload the entire settings structure. However, any changes made with this command will only apply to the working RAM settings, and will be lost upon power-down unless they are subsequently with the CID_SETTINGS_SAVE command.

As an alternative to this command, use CID_SETTINGS_SET.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_SET) |
| AHRS_CAL | AHRSCAL_T | The structure containing the new calibration coefficients for the AHRS system. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_AHRS_CAL_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The command was performed successfully.<br>• CST_CMD_PARAM_MISSING – One of the calibration parameters was not specified correctly. |

## 7.6. Acoustic Transceiver Messages

Acoustic transceiver messages are used to gain information about the activity of the beacons transmitter and receiver systems, independently to how they are being controlled by any message protocol handlers.



With the exception of the CID_XCVR_ANALYSE message, all other messages are generated only when the appropriate enabling flag is set in the acoustic transceiver settings – see CID_SETTINGS_SET.



Regardless of weather acoustic transceiver messages are generated, protocol handlers may generate their own specific status and information messages that may duplicate (or omit) some of the information contained in the messages below. Further details can be found in the definitions of protocol handler messages found in the subsequent sections of this document.

### 7.6.1. CID_XCVR_ANALYSE

The CID_XCVR_ANALYSE command is sent to instruct the transceiver to perform a background noise analysis of the receiver, and report its findings.
On receiving the command, the transceiver will stop any reception activity in progress, then listen to the main receiver transducer for 0.5 seconds to determine noise levels. Once the test in finished, the transceiver will return to the Idle state waiting for a new reception to commence.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_ANALYSE) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_ANALYSE) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The command was performed successfully. |
| • CST_FAIL | | An error has occurred (such as Transmission in progress) that prevented the noise analysis from completing. |
| ADC_MEAN | INT16 | The average reading seen by the receiving analogue-to-digital converter.<br>This value is used primarily for factory diagnostics and commissioning of the beacon hardware. |

| | | |
|---|---|---|
| ADC_PKPK | UINT16 | The peak-to-peak reading seen by the receiving analogue-to-digital converter.<br>This value is used primarily for factory diagnostics and commissioning of the beacon hardware. |
| ADC_RMS | UINT32 | The RMS reading seen by the receiving analogue-to-digital converter.<br>This value is used primarily for factory diagnostics and commissioning of the beacon hardware. |
| RX_LEVEL_PKPK | INT16 | The peak-to-peak noise level observed on the receiver, encoded in centibels. To decode, divide this value by 10 for decibels to a 0.1 dB resolution.<br><br>This is useful to identify is any short bursts of interference are present during the analysis period.<br>Typical values should lie between 85.0dB and 100.0dB for normal operation. |
| RX_LEVEL_RMS | INT16 | The RMS noise level observed on the receiver, encoded in centibels. To decode, divide this value by 10 for decibels to a 0.1 dB resolution.<br><br>This is useful to determine what the general background ambient noise level is. Higher values indicate it is hardware for the receiver to detect valid beacon signals over the noise.<br>Typical values should lie between 80.0dB and 95.0dB for normal operation. |

### 7.6.2. CID_XCVR_TX_MSG

The CID_XCVR_TX_MSG status message is generated when the acoustic transceiver is instructed to send a message to another beacon.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on protocol handlers or acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_TX_MSG) |
| ACO_MSG | ACOMSG_T | Structure populated with the transmitted message details. |

### 7.6.3. CID_XCVR_RX_ERR

The CID_XCVR_RX_ERR status message is generated when the acoustic transceiver has encountered an error while trying to receive an acoustic message, or wait for a request response to be received.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.



Faint degraded or noisy acoustic signals may lead to data corruption on the receiver, which is the most common cause of CST_XCVR_CSUM_ERROR messages being generated.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_ERR) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_XCVR_ID_REJECTED★ | | The transceiver has received a message with a destination ID that does not match its own so the message will be rejected (this is not technically an error, just issued for information). |
| • CST_XCVR_CSUM_ERROR★ | | The transceiver has received a message, but the message checksum CRC16 value does not match the decoded data indicating the message is corrupt and will be discarded. |
| • CST_XCVR_LENGTH_ERROR★ | | The transceiver has started decoding a message, but not reached the end of it within the expected period of time. The message is assumed to be corrupt and will be discarded. |
| • CST_XCVR_RESP_TIMEOUT❖ | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG❖ | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR❖ | | A general reception error has occurred while waiting for a response. |
| • CST_FAIL★ | | An unknown error has occurred. |

| | | |
|---|---|---|
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the error.<br><br>As errors usually occur before a message completes reception, the Flags field won't indicate valid range, USBL signal or position.<br><br>For errors above marked with "❖", no RSSI level will be available, as the error isn't triggered by direct reception of an acoustic message.<br><br>For errors above marked with a "★", a beacon ID for the message sender will not be available, so the SRC_ID/DEST_ID fields will be the local ID of the beacon instead. |

### 7.6.4. CID_XCVR_RX_MSG

The CID_XCVR_RX_MSG status message is generated when the acoustic transceiver has received a valid general message from another beacon that does not require a reply or acknowledgment. After issuing this status message, the acoustic message is passed to the appropriate protocol handler in the acoustic protocol stack, which may generate further messages.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_MSG) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| ACO_MSG | ACOMSG_T | Structure populated with the received Request message details.<br><br>The MSG_TYPE field value will be MSG_OWAY or MSG_OWAYU. |

### 7.6.5. CID_XCVR_RX_REQ

The CID_XCVR_RX_REQ status message is generated when the acoustic transceiver has received a valid request message from another beacon that will require a reply constructing and transmitting back. After issuing this status message, the acoustic message is passed to the appropriate protocol handler in the acoustic protocol stack, which may generate further messages.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_REQ) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| ACO_MSG | ACOMSG_T | Structure populated with the received Request message details.<br><br>The MSG_TYPE field value will be MSG_REQ, MSG_REQU or MSG_REQX. |

### 7.6.6. CID_XCVR_RX_RESP

The CID_XCVR_RX_RESP status message is generated when the acoustic transceiver has received a valid response to a request message it transmitted earlier. After issuing this status message, the acoustic message is passed to the appropriate protocol handler in the acoustic protocol stack, which may generate further messages.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_RESP) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| ACO_MSG | ACOMSG_T | Structure populated with the received Response message details.<br><br>The MSG_TYPE field value will be MSG_RESP, MSG_RESPU or MSG_RESPX. |

### 7.6.7. CID_XCVR_RX_UNHANDLED

The CID_XCVR_RX_UNHANDLED status message is generated when the acoustic transceiver has received a message with a payload protocol identifier that it does not know how to handle within its acoustic protocol stack.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_DIAG_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_UNHANDLED) |
| ACO_MSG | ACOMSG_T | Structure populated with the received message details. |

### 7.6.8. CID_XCVR_USBL

The CID_XCVR_USBL status message is generated when the acoustic transceiver has received a message that contains a USBL signal, from which incoming signal angle information can be computed. The message contents are design to assist with diagnosing the validity of acoustic signals, and debug causes of reception/decoding failure.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_USBL_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.



The CID_XCVR_USBL message is large in size (typically 461 bytes) so suitable memory buffering should be allocated when decoding.
Sending this message over a slow serial communications link may degrade the performance of the beacon.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_USBL) |
| XCOR_SIG_PEAK | FLOAT | The magnitude of the peak signal observer while correlating the USBL signal. |
| XCOR_THRESHOLD | FLOAT | The magnitude of the detection threshold being used to detect the first peak in the correlated USBL signal. |
| XCOR_CROSS_POINT | UINT16 | The correlation sample number where the USBL signal passes the detection threshold. |
| XCOR_CROSS_MAG | FLOAT | The actual magnitude of the correlated signal when the USBL signal passed the detection threshold. |
| XCOR_DETECT | UINT16 | The correlation sample number where the first peak of USBL signal was found. |
| XCOR_LENGTH | UINT16 | The number of correlation samples computed for the received signal. Typically this value is 100. |
| XCOR_DATA | FLOAT[x] | Array of floating point data containing the correlated received signal, where “x” is equal to the value defined in XCOR_LENGTH. |
| CHANNELS | UINT8 | The number of USBL receiver channels being used to compute the signal angle. Typically this value is either 3 or 4. |

| | | |
|---|---|---|
| CHANNEL_RSSI | INT16[x] | An array of the received signal strengths for each of the USBL receiver channels, where “x” is the value defined by the CHANNELS field.<br>Values are encoded in centi-Bels, so divide by 10 to obtain a value in decibels to a resolution of 0.1dB. |
| BASELINES | UINT8 | The number of baselines available from which position information can be computed. Typically this value is either 3 or 6. |
| PHASE_ANGLE | FLOAT[x] | An array of the measured phase angles for each baseline (between pairs of USBL receiver elements), where “x” is the value defined by the BASELINES field. |
| SIGNAL_AZIMUTH | INT16 | The incoming signal azimuth angle from 0° to 360°.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| SIGNAL_ELEVATION | INT16 | The incoming signal elevation angle from -90° to +90°.<br>Values are encoded as deci-Degrees, so divide by 10 for just degrees to a 0.1° resolution. |
| SIGNAL_FIT_ERROR | FLOAT | The fit error value returns a number that indicates the quality of fit (or confidence) of the signal azimuth and elevation values from the timing and phase-angle data available.<br><br>Smaller values towards 0.0 indicate a better fit, while larger values (increasing above 2-3) indicate poorer fits and larger error tolerances. |
| BEACON_DEST_ID | BID_E | Identifier for the beacon the message is or was the recipient of the transmission. |
| BEACON_SRC_ID | BID_E | Identifier for the beacon that generated or sent the transmission. |

### 7.6.9. CID_XCVR_FIX

The CID_XCVR_FIX status message is generated when the acoustic transceiver has received a message from which information about the remote beacons position can be determined.

If the received acoustic message MSG_TYPE field is either MSG_OWAYU, MSG_RESPU or MSG_RESPX, then the received USBL signal information can be used to compute the incoming angle of the message.

In the case of MSG_RESPU and MSG_RESPX messages, these are sent in response to a request message, and because round-trip timing is then available a range can be determined to the remote beacon, and combined with incoming signal angles, a relative position estimate can be made.
However, for MSG_OWAYU messages no such timing information in available, so only incoming signal angle information is available in the fix message.

Messages with the MSG_RESPX type also contain an additional field specifying the depth of the remote beacon. From this information an enhanced vertical position fix can be computed.

For message types of MSG_RESP, no USBL signal information is available, so only ranging information can be computed.

Messages types of MSG_OWAY do not provide USBL information, or allow for timing to be performed, so Fix messages will not be generated on their reception.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_FIX_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_RX_FIX) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |

### 7.6.10. CID_XCVR_STATUS

The CID_XCVR_STATUS command can be used to determine the current operating state of the acoustic transceiver modules.
It is particularly useful to use prior to issuing other commands that may start transmissions (i.e. CID_PING_SEND, CID_ECHO_SEND etc.) to check that the transceiver is currently in the IDLE state and can perform the operation.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_STATUS) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_STATUS) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_XCVR_STATE_STOPPED | | Indicates the transceiver is in a stopped state. |
| • CST_XCVR_STATE_IDLE | | Indicates the transceiver is in an idle state waiting for reception or transmission to start. |
| • CST_XCVR_STATE_TX | | Indicates the transceiver is in a transmitting states. |
| • CST_XCVR_STATE_REQ | | Indicates the transceiver is in a requesting state, having transmitted a message and is waiting for a response to be received. |
| • CST_XCVR_STATE_RX | | Indicates the transceiver is in a receiving state. |
| • CST_XCVR_STATE_RESP | | Indicates the transceiver is in a responding state, where a message is being composed and the “response time” period is being observed. |

### 7.6.11. CID_XCVR_TX_MSGCTRL_SET

The CID_XCVR_TX_MSGCTRL_SET command is used to directly modify and apply the XCVR_TX_MSGCTRL control bits in the XCVR_FLAGS settings register of the SETTINGS_T structure.

Conceptually, these bits can act like the “airplane” mode control on a mobile phone, preventing automated transmission of response acoustic messages, or all acoustic messages. In such cases, all other actions/outputs will still be performed by the beacon, only transmission will be prevented.

The current value of the TX_MSGCTRL bits can be found by querying settings using CID_SETTINGS_GET or sending this command with a value of 0xFF (or omitted completely).

If the settings structure is stored into flash with CID_SETTINGS_SAVE after this command is issued, then the status of the TX_MSGCTRL flags will be stored and applied on subsequent power-up of the beacon.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_TX_MSGCTRL_SET) |
| TXMSGCTRL | XCVR_TXMSGCTRL_E | Uint8 numerical value of a XCVR_TXMSGCTRL_E enumeration specifying the Transmit Message Control mode of the acoustic transceiver.<br>Not specifying this parameter will allow a query on the current value to be performed, but a CST_CMD_PARAM_MISSIG error code will be returned.<br>Specifying a value of 0xFF will allow the value to be queried, (with no change to settings made) and CST_OK will be returned. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_TX_MSGCTRL_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The command was performed successfully. |
| • CST_CMD_PARAM_MISSING | | The TXMSGCTRL value has not been specified correctly. |

| | | |
|---|---|---|
| TXMSGCTRL | XCVR_TXMSGCTRL_E | A confirmation of the new TXMSGCTRL value, or a query of the current value. |

### 7.6.12. CID_XCVR_USBL_TIME

The CID_XCVR_USBL_TIME status message is generated when the acoustic transceiver has received a message that contains a USBL signal, from which incoming signal angle information can be computed. The message provides the time delta between the USBL signal being received, and this message being sent. This is useful for systems using an external heading source as the heading at the time of receiving the USBL signal can be more accurately captured.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_BASELINES) |
| TIME_DELTA | UINT16 | The number of milliseconds between the USBL signal being received, and this message being transmitted out of the beacon. |

### 7.6.14. CID_XCVR_BASELINES

The CID_XCVR_BASELINES status message is generated when the acoustic transceiver has received a message that contains a USBL signal, from which incoming signal angle information can be computed. The message contents are designed to provide the required information to perform the signal angle calculations and compute a position.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



This message is generated only when the XCVR_BASELINES_MSGS flag is specified in the acoustic transceiver SETTINGS_T structure – see CID_SETTINGS_SET.



Enabling this message will disable the internal signal angle and position calculations. In order to get a position with this message enabled the SDK must be used. This can help to speed up the system as the calculations can be performed faster by a PC than by the microcontroller in the beacon.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_XCVR_BASELINES) |
| BEACON_SRC_ID | BID_E | The ID of the beacon that sent the acoustic USBL message from which the baselines in this message were calculated. |
| BASELINE_COUNT | UINT8 | The number of baselines available from which position information can be computed. Typically this value is either 3 or 6. |
| BASELINES | FLOAT[x] | An array of the measured phase angles (in radians) for each baseline (between pairs of USBL receiver elements), where “x” is the value defined by the BASELINE_COUNT field. |

# 8. Acoustic Protocol Stack Message Definitions

## 8.1. PING Protocol Messages

The acoustic PING protocol provides a set of commands and status messages that can be used to determine the presence of other beacons and obtain range and position values for the remote “pinged” beacon.

Unlike other protocols, PING messages do not carry any payload information making them the shortest messages possible (except TRACK which works a little differently). This in turn leads to least power consumption for battery powered operations, and the fastest polling speed in networked or navigation systems.

Once the beacon is correctly configured, a PING operation is started by sending the CID_PING_SEND command.

### 8.1.1. CID_PING_SEND

The CID_PING_SEND command is issued to perform an acoustic “ping” on another beacon, from which its range and position can be determined.

For positioning, acoustic PING messages are the shortest of all the protocols available (except TRACK which works a little differently), requiring the least power consumption on the transmitter and allowing the fastest operation when polling round a network.



If successful, the CID_PING_SEND command will start an acoustic message transmission, and at some time later an acoustic response will be heard.
The PING protocol handler will generate subsequent serial message to signal when these events occur. Additionally, acoustic transceiver diagnostic messages are available to monitor activity and obtain further information on the received signal levels – see section 7.6 on page 108 for further details.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PING_SEND) |
| DEST_ID | BID_E | The ID code of the beacon to send the Ping to, and receive position and ranging information for.<br>Valid values are form 1 to 100. |
| MSG_TYPE | AMSGTYPE_E | Value specifying the type of data message that should be sent<br>Valid values are… |
| • MSG_REQ | | Data is sent as a request message, but no USBL information is required.<br>As no USBL signal information is transmitted, but a response is returned, ranging information will be available for the remote beacon. |
| • MSG_REQU | | Data is sent as a request message and a USBL signal is required.<br>As both USBL signal information and a response is returned by the remote beacon, positioning and range information for the remote beacon will be available to the sender. |
| • MSG_REQX | | As MSG_REQU but an enhanced USBL request is sent (i.e. remote depth sensor information included to enhance the depth position fix). |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PING_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The PING command is being sent. |
| • CST_CMD_PARAM_INVALID | | The DEST_ID parameter is invalid. |
| • CST_CMD_PARAM_MISSING | | The DEST_ID has not been specified correctly. |
| • CST_XCVR_BUSY | | The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| BEACON_ID | BID_E | The ID code of the beacon that the command was sent to.<br>Valid values are form 1 to 100. |

### 8.1.2. CID_PING_REQ

The CID_PING_REQ status message is generated when the beacon received an acoustic PING message from the sending beacon. This message is provided by the protocol-handler in the beacon for information purposes only, and no action or acknowledgment by the external system/user is required.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PING_REQ) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to signal from the beacon sending data.<br>This will not contain range or position information. |

### 8.1.3. CID_PING_RESP

The CID_PING_RESP status message is generated by the sending beacon when it receives a PING response back from the remote (“pinged”) beacon.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PING_RESP) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |

### 8.1.4. CID_PING_ERROR

The CID_PING_ERROR status message is generated when a PING operation is not successful, usually because a response is not received from the remote interrogated beacon, causing a timeout.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PING_ERROR) |
| STATUS | CST_E | Status code indicating the error that occurred.<br>Valid values are… |
| • CST_XCVR_RESP_TIMEOUT | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR | | A general reception error has occurred while waiting for a response. |
| BEACON_ID | BID_E | The ID code of the beacon that the error applies to (i.e. the DEST_ID value used in the CID_PING_SEND command).<br>Valid values are form 1 to 100. |

## 8.2. ECHO Protocol Messages

The ECHO protocol is primarily intended for testing and validating remote beacons functionality and diagnosing problems with acoustic communications.

The CID_ECHO_SEND command is used to start an ECHO transmission, where a user specified payload of up to 31 bytes is sent to the remote beacon. Upon successful reception, the remote beacon will then return a response message containing the same payload.

While the acoustic transceiver should receive the response message correctly, and not have to validate the contents of the response data (guaranteed by the CRC checksum algorithms used), the longer structures used by the ECHO protocol over the PING protocol allow acoustic channel characteristics to be evaluated or the effect of Doppler shift on remote beacons.

### 8.2.1. CID_ECHO_SEND

The CID_ECHO_SEND command is issued to send a packet of data to another remote beacon, and have this beacon return (or echo) the data back, also allowing its range and position to be determined.



If successful, the CID_ECHO_SEND command will start an acoustic message transmission, and at some time later an acoustic response will be heard.
The ECHO protocol handler will generate subsequent serial message to signal when these event occur. Additionally, acoustic transceiver diagnostic messages are available to monitor activity and obtain further information on the received signal levels – see section 7.6 on page 108 for further details.



When sending ECHO data, be aware that larger quantities of data will take longer to send (by 80ms per byte), and during this time the receiver may become more susceptible to errors caused by Doppler shift or other interference, and the statistical probability of the CRC16 checksum algorithm (used by the acoustic transceiver) uniquely validating the message content becomes reduced.

When testing the acoustic link, it is recommended to use data lengths of 16 bytes or less if problems with communications are observer.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_SEND) |
| DEST_ID | BID_E | The ID code of the beacon to send the Echo request to, and receive position and ranging information for.<br>Valid values are form 1 to 100. |
| MSG_TYPE | AMSGTYPE_E | Value specifying the type of data message that should be sent |

| | | |
|---|---|---|
| | | Valid values are…<br>• MSG_REQ: Data is sent as a request message, but no USBL information is required. As no USBL signal information is transmitted, but a response is returned, ranging information will be available for the remote beacon.<br>• MSG_REQU: Data is sent as a request message and a USBL signal is required. As both USBL signal information and a response is returned by the remote beacon, positioning and range information for the remote beacon will be available to the sender.<br>• MSG_REQX: As MSG_REQU but an enhanced USBL request is sent (i.e. remote depth sensor information included to enhance the depth position fix). |
| PACKET_LEN | UINT8 | The number of bytes to send in the Echo acoustic packet. Valid values are from 0 to 30. |
| PACKET_DATA | UINT8[x] | The array of data that should be sent in the Echo acoustic packet, where “x” is the value specified in PACKET_LEN. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK: The ECHO command is being sent.<br>• CST_CMD_PARAM_INVALID: The DEST_ID parameter is invalid.<br>• CST_CMD_PARAM_MISSING: The DEST_ID has not been specified correctly.<br>• CST_XCVR_BUSY: The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| BEACON_ID | BID_E | The ID code of the beacon that the command was sent to. Valid values are form 1 to 100. |

### 8.2.2. CID_ECHO_REQ

The CID_ECHO_REQ status message is generated when the beacon received an acoustic ECHO message from the sending beacon. This message is provided by the protocol-handler in the beacon for information purposes only, and no action or acknowledgment by the external system/user is required.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_REQ) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to signal from the beacon sending data.<br>This will not contain range or position information. |
| PACKET_LEN | UINT8 | The number of bytes sent in the Echo acoustic packet. Valid values are from 0 to 30. |
| PACKET_DATA | UINT8[x] | The array of data received in the Echo acoustic packet, where “x” is the value specified in ECHO_LEN. |

### 8.2.3. CID_ECHO_RESP

The CID_ECHO_RESP status message is generated by the sending beacon when it receives an ECHO response back from the remote beacon. The payload of the response message should carry a duplicate copy of the data that was transmitted with the CID_ECHO_SEND command.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_RESP) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| PACKET_LEN | UINT8 | The number of bytes received back from the remote beacon in the Echo acoustic packet.<br>Valid values are from 0 to 30. |
| PACKET_DATA | UINT8[x] | The array of data received in the Echo acoustic packet, where “x” is the value specified in PACKET_LEN. |

### 8.2.4. CID_ECHO_ERROR

The CID_ECHO_ERROR status message is generated when an ECHO operation is not successful, usually because a response is not received from the remote interrogated beacon, causing a timeout.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_ERROR) |
| STATUS | CST_E | Status code indicating the error that occurred.<br>Valid values are… |
| • CST_XCVR_RESP_TIMEOUT | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR | | A general reception error has occurred while waiting for a response. |
| BEACON_ID | BID_E | The ID code of the beacon that the error applies to (i.e. the DEST_ID value used in the CID_ECHO_SEND command).<br>Valid values are form 1 to 100. |

## 8.3. DAT Protocol Messages

The DAT (or datagram) protocol forms the basis for any data exchange between pairs of SeaTrac Beacons.

When the controlling system or user, issues a CID_DAT_SEND command, a packet of up to 31 bytes is transmitted to the specified beacon.
Optionally the user may also choose if they wish to receive a response back from the remote beacon. This allows acknowledgment of reception to occur, and additionally will include any data that may have been queued onto a packet buffer at the remote end of the link (see CID_DAT_QUEUE_SET) – allowing simple bidirectional data transfers to occur.

The DAT protocol is deliberately basic in its implementation, allowing system developers to decide how to format and interpret the data packets – for example implementing "port" numbers (similar to the UDP protocol), sequence pointers etc.

By implementing a simple controlling state machine, developers can use the ACK/ACK_USBL send modes to determine the presence of the remote beacon and remotely retrieve data. CID_DAT_RECEIVE messages will be returned to indicate data has been received at either end of the link, or CID_DAT_ERROR messages to indicate undelivered data in the case of a timeout or other reception error occurring.

At either receiving end, data is not buffered, but output with a CID_DAT_RECEIVE message as it arrived, again allowing system developers to handle the data in the best way for their application.



When sending DAT data, be aware that larger quantities of data will take longer to send (by 80ms per byte), and during this time the receiver may become more susceptible to errors caused by Doppler shift or other interference, and the statistical probability of the CRC16 checksum algorithm (used by the acoustic transceiver) uniquely validating the message content becomes reduced.

Where possible it is recommended (but not required) to limit packet lengths to between 16 to 20 bytes.



As mentioned above, the simplicity of the DAT protocol also leads to limitations if trying to send larger amounts of data (across several packets) or verify successful reception of the data in sequence, resending missing packets.



The DAT protocol supports 'packet sniffing' where the SeaTrac beacon will intercept and report every data messages receives through the CID_DAT_RECEIVE output message, even if the message is not intended for the beacon.
The application developer should use the SRC_ID and DEST_ID in the ACO_FIX structure to determine if the data message should be used.

### 8.3.1. CID_DAT_SEND

The CID_DAT_SEND command is issued to send a packet of data to another remote beacon, optionally requesting that the remote beacon acknowledge reception of the data.

If the command requires an acknowledgement (ACK), then at some point later upon reception of the response message a CID_DAT_RECEIVE message will be output containing the ACK flag and any additional remotely queued data that has been transmitted back.



If successful, the CID_DAT_SEND command will start an acoustic message transmission, and at some time later an acoustic response will be heard.
The DAT protocol handler will generate subsequent serial message to signal when these events occur. Additionally, acoustic transceiver diagnostic messages are available to monitor activity and obtain further information on the received signal levels – see section 7.6 on page 108 for further details.



When sending DAT data, be aware that larger quantities of data will take longer to send (by 80ms per byte), and during this time the receiver may become more susceptible to errors caused by Doppler shift or other interference, and the statistical probability of the CRC16 checksum algorithm (used by the acoustic transceiver) uniquely validating the message content becomes reduced.

Where possible it is recommended (but not required) to limit packet lengths to between 16 to 20 bytes.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_SEND) |
| DEST_ID | BID_E | The ID code of the beacon to send the DATA to.<br>Valid values are from 0 to 100.<br><br>A value of BEACON_ALL (0) indicates data is broadcast to all other beacons. However when this is used the SEND_MODE value can only be NOACK or NACK_USBL. |
| MSG_TYPE | AMSGTYPE_E | Value specifying the type of data message that should be sent<br>Valid values are… |
| • MSG_OWAY | | Data is sent one-way with no USBL signal request, and no response acknowledgment is required.<br>As no USBL signal or response is used, no ranging or position fix will be available on receipt on this message. |
| • MSG_OWAYU | | Data is sent one-way with a USBL signal request, but no response acknowledgment is required. |

| | | |
|---|---|---|
| | | As a USBL signal is transmitted with the data, the remote beacon can determine incoming signal angle, but not range or position. |
| | • MSG_REQ | Data is sent as a request message where the acknowledgment will be the response, but no USBL response is required.<br>As no USBL signal information is transmitted, but a response is returned, ranging information will be available for the remote beacon. |
| | • MSG_REQU | Data is sent as a request message where the acknowledgment will be the response, and a USBL signal is required.<br>As both USBL signal information and a response is returned by the remote beacon, positioning and range information for the remote beacon will be available to the sender. |
| | • MSG_REQX | As MSG_REQU but an enhanced USBL request is sent. |
| PACKET_LEN | UINT8 | The number of bytes to send in the DAT acoustic packet. Valid values are from 0 to 31. |
| PACKET_DATA | UINT8[x] | The array of data that should be sent in the DAT acoustic packet, where “x” is the value specified in PACKET_LEN. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| | • CST_OK | The DAT command is being sent. |
| | • CST_CMD_PARAM_INVALID | The value of one of the parameters is invalid. |
| | • CST_CMD_PARAM_MISSING | There is not enough data parameters provided to satisfy the message requirements. |
| | • CST_XCVR_BUSY | The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| BEACON_ID | BID_E | The ID code of the beacon that the command was sent to.<br>Valid values are from 1 to 100. |

### 8.3.2. CID_DAT_RECEIVE

The CID_DAT_RECEIVE status message is generated when the beacon received a DAT protocol data packet from another beacon.

Additionally, this message will be issued upon reception of an acknowledgement response back from a remote beacon if the CID_DAT_SEND command specified its MSG_TYPE value to be either MSG_RESP, MSG_RESPU or MSG_RESPX. In this case, if the remote beacon had any data queued, it will also have been transmitted and is output with this message.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



The receiving beacon (generating this message) will automatically handle generation of requested ACK messages if required. Beyond handling the data in the reported packet not further interaction is required with the external system/user.



The DAT protocol supports 'packet sniffing', and will generate CID_DAT_RECEIVE messages for EVERY acoustic message a beacon intercepts, even if the message is not intended for the beacon.
The application developer should use the LOCAL_FLAG field to determine if the data message should be used.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_RECEIVE) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data.<br>Messages generated at the remote beacon (or MSG_OWAY/MSG_OWAYU types were used) will not contain range or position information. |
| ACK_FLAG | BOOLEAN | Flag is true if this message has been generated as a response to a CID_DAT_SEND command which requested an ACK – in which case, remotely queued data may have also been transmitted back and in included in this message. |
| PACKET_LEN | UINT8 | The number of bytes sent in the DAT acoustic packet. Valid values are from 0 to 31.<br>A value of 0 indicate no data is present. |
| PACKET_DATA | UINT8[x] | The array of data received in the DAT acoustic packet, where "x" is the value specified in PACKET_LEN. |
| LOCAL_FLAG | BOOLEAN | True if an acoustic DAT message has been received that is address for this local beacon (or a broadcast to all). |

This flag is the same as the Boolean result of logical test “ACO_FIX.DEST_ID = LOCAL_BEACON_ID or BEACON_ALL”

False, if the received request message is intended for another beacon, and has been ‘sniffed’ by the receiver. This gives the application developer the option optimally process and synchronise shared data across the network sent in the PACKET_DATA field.

### 8.3.3. CID_DAT_ERROR

The CID_DAT_ERROR status message is generated when a DAT operation is not successful, usually because a response is not received from the remote interrogated beacon, causing a timeout.

This message will only be generated if the CID_DAT_SEND command specified its MSG_TYPE field to be MSG_REQ, MSG_REQU or MSG_REQX, as only these modes allow the remote beacon to report back its status.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_ERROR) |
| STATUS | CST_E | Status code indicating the error that occurred.<br>Valid values are… |
| • CST_XCVR_RESP_TIMEOUT | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR | | A general reception error has occurred while waiting for a response. |
| BEACON_ID | BID_E | The ID code of the beacon that the error applies to (i.e. the DEST_ID value used in the CID_DAT_SEND command).<br>Valid values are form 1 to 100. |

### 8.3.4. CID_DAT_QUEUE_SET

The CID_DAT_QUEUE_SET command is used to store (queue) a single packet of data in the beacon that will be sent to the specified destination beacon when requested by it (using an acoustic CID_DAT_SEND message).



The DAT queue only holds one packet for each specific beacon address. Subsequent CID_DATA_QUEUE_SET commands will overwrite any previously un-transmitted data for that destination beacon.

Specifying a PACKET_LEN of 0 will clear down any pending messages on the relevant queue, or all queues for DEST_ID = BEACON_ALL (0).

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_SET) |
| DEST_ID | BID_E | The ID code of the beacon to that the queued packet will be sent to in response to a CID_DATA_SEND command. Requests received from other beacons will be ignored.<br>Valid values are from 0 to 100.<br><br>A value of BEACON_ALL (0) indicates data is queued for all destinations.<br><br>Data cannot be queued for the same ID as that used by the beacon itself. |
| PACKET_LEN | UINT8 | The optional number of bytes to queue.<br>Valid values are from 0 to 31.<br>Using a value of 0 will clear the contents of the specified queue(s).<br><br>Omitting this value allows the length of the queue to be queried in the response and will NOT affect the contents of the queue. |
| PACKET_DATA | UINT8[x] | If PACKET_LEN was specified, this contains the array of data that should be sent in the DAT acoustic packet, where “x” is the value specified in PACKET_LEN.<br><br>If PACKET_LEN was not sent, this data should not be sent either. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK — The packet has been queued.<br>• CST_CMD_PARAM_INVALID — The value of one of the parameters is invalid.<br>• CST_CMD_PARAM_MISSING — There are not enough data parameters provided to satisfy the message requirements. |
| DEST_ID | BID_E | The beacon ID code of the queue that has been modified/queried.<br>Valid values are from 0 to 100. |
| PACKET_LEN | UINT8 | The length of the packet in the queue for the specified DEST_ID beacon.<br>If DEST_ID was BEACON_ALL (0), then this will return the new packet length loaded, unless PACKET_LEN was omitted then 0 is returned. |

### 8.3.5. CID_DAT_QUEUE_CLR

The CID_DAT_QUEUE_CLR command is used to clear any packed that has been queued for remote retrieval by the DAT protocol, using the CID_DATA_QUEUE_SET command.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_CLR) |
| DEST_ID | BID_E | Optional ID code of the beacon to that the queued packet will be sent to in response to a CID_DATA_SEND command.<br>Requests received from other beacons will be ignored.<br>Valid values are from 0 to 100.<br><br>A value of BEACON_ALL (0) indicates data is queued for all destinations. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_CLR) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The DAT command is being sent. |
| DEST_ID | BID_E | The beacon ID code of the queue that has been cleared.<br>Valid values are from 0 to 100. |

### 8.3.6. CID_DAT_QUEUE_STATUS

The CID_DAT_QUEUE_STATUS command is used to determine if any packet is queued and waiting for remote retrieval by the DAT protocol in response to a CID_DAT_SEND command.

The number of bytes in the packet payload is returned in response to this command, and when the packet has been delivered, this value is reported as 0, indicating empty (and another packet can be queued).

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_STATUS) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_STATUS) |
| PACKET_LEN | UINT8[101] | Array of 101 lengths representing the pending number of data bytes queued for each beacon.<br>Valid values for each entry are from 0 to 31, except for PACKET_LEN[0] will always return 0.<br>A value of 0 indicates there is no queued data. |

### 8.3.7. CID_DAT_QUEUE_GET

The CID_DAT_QUEUE_GET command is used to retrieve the packet contents that has been queued up to send to a remote beacon.

The number of bytes in the packet payload, as well as the payload itself is returned in response to this command, a value of zero means that there was no queued data for the beacon.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_DAT_QUEUE_GET) |
| DEST_ID | BID_E | The ID code of the beacon whose queued packet is being retrieved.<br>Valid values are from 1 to 100.<br><br>Data cannot be retrieved for the same ID as that used by the beacon itself. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_GET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The queued packet has been retrieved. |
| • CST_CMD_PARAM_INVALID | | The value of one of the parameters is invalid. |
| • CST_CMD_PARAM_MISSING | | There are not enough data parameters provided to satisfy the message requirements. |
| • DEST_ID | | The beacon ID code of the queue that has been retrieved.<br>Valid values are from 1 to 100. |
| PACKET_LEN | UINT8 | The number of bytes in the queued packet. |
| PACKET_DATA | UINT8[x] | The array of data that has been queued, where “x” is the value specified in PACKET_LEN. |

## 8.4. NAV Protocol Messages

The Navigation protocol builds on the basic range and positioning information produced by the acoustic transceiver, to provide a set of commands useful for incorporating into tracking and navigation systems for ROV's, AUV's or Divers.

As standard, the NAV protocol uses enhanced USBL fixes where the remote beacons depth is transmitted in the response message. From this, USBL beacons can improve the computed vertical position solution.

The NAV protocol is designed to be primarily used in a 'round-robin' scheme, where a single X150 beacon polls all the other beacons in the network using a CID_NAV_QUERY_SEND command. When the response from each beacon is received a CID_NAV_STATUS_SEND command could be used to broadcast the computed position of that beacon to all other beacons in the network (so every beacon will know the position of every other beacon).

The NAV protocol also supports data exchange for messaging type applications – data sent using the CID_NAV_QUERY_SEND command will always be sent to receiving beacons if present, but an optional QRY_DATA/ QRY_DATA_PLUS flag can be set specifying that any queued data on the remote beacon is returned (with packet queuing operating in a similar fashion to the DAT protocols).
However, if QRY_DATA is used and remote data is present, then this will be sent back instead of any other queried data specified in the sent message (i.e. attitude, heading, supply voltage etc).

As with the DAT protocol, the NAV protocol also supports message 'sniffing' where the beacon will process and report acoustic received messages sent between other beacons.
Both the CID_NAV_QUERY_REQ and CID_NAV_QUERY_RESP messages contain a LOCAL_FLAG field which will be true when the received message is addressed for that beacon, otherwise it will be false.
This feature allows the system designer an easy mechanism to ensure that beacons can synchronise their data easily without excessive acoustic messages being required – i.e. if one beacon performs a NAV_QUERY of another beacon's heading information, potentially all other beacons will 'hear' the response and report the requested heading response with a CID_NAV_QUERY_RESP message.

### 8.4.1. CID_NAV_QUERY_SEND

The CID_NAV_QUERY_SEND command is issued to request the specified remote beacon respond with the requested information. Additionally, data can be sent to the remote beacon and queued data requested back from the remote beacon if available.

Regardless of which information if requested, the response contains an enhanced USBL message, where the remote depth sensor reading is encoded to a resolution of 1m, and this is used to compute the vertical Z-axis depth from surface of the position fix. The only exception to this is when remote Depth information is requested, then this value (to 0.1m resolution) is used for the Z-axis depth fix. Note that in circumstances where depth information is requested in the QUERY_FLAGS field but is not provided in the response (this can happen if data is also requested using either QRY_DATA or QRY_DATA_PLUS flag), the protocol will fall back to the enhanced USBL message with 1m resolution depth reading.



If successful, the CID_NAV_QUERY_SEND command will start an acoustic message transmission, and at some time later an acoustic response will be heard (or a timeout happens).
The NAV protocol handler will generate subsequent serial messages to signal when these events occur. Additionally, acoustic transceiver diagnostic messages are available to monitor activity and obtain further information on the received signal levels – see section 7.6 on page 108 for further details.



By default, the NAV protocol uses an enhanced USBL fix that returns the remote beacons depth to a resolution of 1m, and this is used to augment the position fix.
However, if the QRY_DEPTH bit is specified in the QUERY_FLAGS field, the remote beacon will return the depth to a resolution of 0.1m and use this value instead in the position fix as well as returning it in response message REMOTE_DEPTH field and FIX structure.



When requesting back remote queued data using the QRY_DATA flag in QUERY_FIELDS, other queried parameters cannot be sent at the same time due to the constraining size of the acoustic packet. Instead, returned data (if available) will always take precedence over other query flags (such as attitude, supply voltage etc), which will be returned as FALSE in the CID_NAV_QUERY_RESP message. However, if no data is available, then the other requested information will be returned.



When requesting back remote queued data using the QRY_DATA_PLUS flag in QUERY_FIELDS, other queried parameters may still be sent at the same time. If there is data, this will always be included in the response, and if there is space in the payload, the other queried fields will be included as far as they will fit.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|

| | | |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUERY_SEND) |
| DEST_ID | BID_E | The ID code of the beacon to send the Echo request to, and receive position and ranging information for.<br>Valid values are from 1 to 100. |
| QUERY_FLAGS | NAV_QUERY_T | Bit mask that contains the fields the CID_NAV_QUERY_RESP status message should return. |
| PACKET_LEN | UINT8 | Optional number of bytes to send in the NAV acoustic packet.<br>Valid values are from 0 to **30** (not 31!).<br><br>Omit this parameter (and PACKET_DATA) or use a value of 0 if not required. |
| PACKET_DATA | UINT8[x] | If PACKET_LEN is specified greater than 0, then these fields specify the array of data that should be sent in the NAV acoustic packet, where "x" is the value specified in PACKET_LEN. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUERY_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The ECHO command is being sent.<br>• CST_CMD_PARAM_INVALID – The value of one of the parameters is invalid.<br>• CST_CMD_PARAM_MISSING – The is not enough data parameters provided to satisfy the message requirements.<br>• CST_XCVR_BUSY – The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| DEST_ID | BID_E | The ID code of the beacon that the command was sent to.<br>Valid values are from 1 to 100. |

### 8.4.2. CID_NAV_QUERY_REQ

The CID_NAV_QUERY_REQ status message is generated when the beacon received an acoustic NAV message from the sending beacon. This message is provided by the protocol-handler in the beacon for information purposes only, and no action or acknowledgment by the external system/user is required.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



The NAV protocol supports 'packet sniffing', and will generate CID_NAV_QUERY_REQ messages for EVERY acoustic message a beacon intercepts, even if the message is not intended for the beacon.
The application developer should use the LOCAL_FLAG field to determine if the message was intended for the local beacon (true) or intercepted communications between two other beacons (false).

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUERY_REQ) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| QUERY_FLAGS | NAV_QUERY_T | Bit mask that contains the fields the CID_NAV_QUERY_SEND message specified. |
| PACKET_LEN | UINT8 | The number of bytes sent in the NAV acoustic packet. Valid values are from 0 to **30** (not 31!). |
| PACKET_DATA | UINT8[x] | The array of data received in the NAV acoustic packet, where "x" is the value specified in PACKET_LEN. |
| LOCAL_FLAG | BOOLEAN | True if an acoustic NAV_QUERY request message has been received that is address for this local beacon.<br>This flag is the same as the Boolean result of logical test "ACO_FIX.DEST_ID = LOCAL_BEACON_ID"<br><br>False, if the received request message is intended for another beacon, and has been 'sniffed' by the receiver. This gives the application developer the option to optimally process and synchronise shared data across the network sent in the PACKET_DATA field. |

### 8.4.3. CID_NAV_QUERY_RESP

The CID_NAV_QUERY_RESP status message is generated when the beacon receives back a message from the remote beacon in response to a CID_NAV_QUERY_SEND command.
In the response, the requested remote beacons information specified by the QUERY_FLAGS parameter is encoded, and output via this message.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.



Note: By default, the NAV protocol uses an enhanced USBL fix that returns the remote beacons depth to a resolution of 1m, and this is used to augment the position fix.
However, if the QRY_DEPTH bit is specified in the sent QUERY_FLAGS field, the remote beacon will return the depth to a resolution of 0.1m and use this value instead in the position fix as well as returning it in the REMOTE_DEPTH field and FIX structure.



The NAV protocol supports 'packet sniffing', and will generate CID_NAV_QUERY_RESP messages for EVERY acoustic message a beacon intercepts, even if the message is not intended for the beacon.
The application developer should use the LOCAL_FLAG field to determine if the message was intended for the local beacon (true) or intercepted communications between two other beacons (false).

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUERY_RESP) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data.<br><br>See note above regarding enhanced depth resolution when using the QRY_DEPTH flag. |
| QUERY_FLAGS | NAV_QUERY_T | Bit mask that contains the fields the CID_NAV_QUERY_SEND message specified. |

#### Depth Fields

If the message QUERY_FLAGS parameter contains the QRY_DEPTH bit (see NAV_QUERY_T), then the following fields are sequentially appended to the message record…

| | | |
|---|---|---|
| REMOTE_DEPTH | INT32 | The remote beacons depth based on the measured environmental pressure. |

| | | |
|---|---|---|
| | | Values are encoded in deci-metres, so divide by 10 for a value in metres.<br><br>See note above regarding enhanced depth resolution when using the QRY_DEPTH flag. |

### Supply Fields

If the message QUERY_FLAGS parameter contains the QRY_SUPPLY bit (see NAV_QUERY_T), then the following fields are sequentially appended to the message record…

| | | |
|---|---|---|
| REMOTE_SUPPLY | UINT16 | The remote beacons supply voltage.<br>Values are encoded in milli-volts, so divide by 1000 for a value in Volts. |

### Temperature Fields

If the message QUERY_FLAGS parameter contains the QRY_TEMP bit (see NAV_QUERY_T), then the following fields are sequentially appended to the message record…

| | | |
|---|---|---|
| REMOTE_TEMP | INT16 | The temperature of air/water in contact with the diaphragm of the pressure sensor or the remote beacon.<br>Values are encoded in deci-Celsius, so divide by 10 to obtain a value in Celsius. |

### Attitude Fields

If the message QUERY_FLAGS parameter contains the QRY_ATTITUDE bit (see NAV_QUERY_T), then the following fields are sequentially appended to the message record…

| | | |
|---|---|---|
| REMOTE_YAW | INT16 | The Yaw angle of the remote beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |
| REMOTE_PITCH | INT16 | The Pitch angle of the remote beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |
| REMOTE_ROLL | INT16 | The Roll angle of the remote beacon, relative to magnetic north, measured by the beacons AHRS system.<br>Values are encoded as deci-degrees, so divide the value by 10 to obtain a value in degrees. |

### Data Fields

If the message QUERY_FLAGS parameter contains the QRY_DATA or QRY_DATA_PLUS bit (see NAV_QUERY_T), then the following fields are sequentially appended to the message record.

| | | |
|---|---|---|
| PACKET_LEN | UINT8 | The number of bytes received in the NAV acoustic response packet. Valid values are from 0 to **30** (not 31!). |
| PACKET_DATA | UINT8[x] | The array of data received in the NAV acoustic response packet, where “x” is the value specified in PACKET_LEN. |

### Final Fields

To ensure backward compatibility, the following fields are always present at the end of each message, regardless of previous content.

| | | |
|---|---|---|
| LOCAL_FLAG | BOOLEAN | True if an acoustic NAV_QUERY response message has been received that is address for this local beacon.<br>This flag is the same as the Boolean result of logical test “ACO_FIX.DEST_ID = LOCAL_BEACON_ID”<br><br>False, if the received response message is intended for another beacon, and has been ‘sniffed’ by the receiver. This gives the application developer the option optimally process and synchronise shared data across the network sent in response the the NAV_QUERY flags. |

### 8.4.4. CID_NAV_ERROR

The CID_NAV_ERROR status message is generated when an NAV operation is not successful, usually because a response is not received from the remote interrogated beacon (causing a timeout) or a malformed payload has been decoded.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_ECHO_ERROR) |
| STATUS | CST_E | Status code indicating the error that occurred.<br>Valid values are… |
| • CST_XCVR_RESP_TIMEOUT | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR | | A general reception error has occurred while waiting for a response. |
| • CST_XCVR_PLOAD_ERROR | | The payload of the acoustic message does not contain the expected or required data, and the operation cannot be performed – this can occur on reception of a Request or Response message. |
| BEACON_ID | BID_E | The ID code of the beacon that the error applies to (i.e. the DEST_ID value used in the CID_NAV_QUERY_SEND command).<br>Valid values are form 1 to 100. |

### 8.4.5. CID_NAV_QUEUE_SET

The CID_NAV_QUEUE_SET command is used to store (queue) a single packet of data in the beacon that will be sent to the specified destination beacon when requested by it (using an acoustic CID_NAV_QUERY_SEND message).



The NAV queue only holds one packet for each specific beacon address. Subsequent CID_NAV_QUEUE_SET commands will overwrite any previously un-transmitted data for that destination beacon.

Specifying a PACKET_LEN of 0 will clear down any pending messages on the relevant queue, or all queues for DEST_ID = BEACON_ALL (0).

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_SET) |
| DEST_ID | BID_E | The ID code of the beacon to that the queued packet will be sent to in response to a CID_NAV_QUERY_SEND command. Requests received from other beacons will be ignored. Valid values are from 0 to 100.<br><br>A value of BEACON_ALL (0) indicates data is queued for all destinations.<br><br>Data cannot be queued for the same ID as that used by the beacon itself. |
| PACKET_LEN | UINT8 | The optional number of bytes to queue. Valid values are from 0 to **30**. Using a value of 0 will clear the contents of the specified queue(s).<br><br>Omitting this value allows the length of the queue to be queried in the response and will NOT affect the contents of the queue. |
| PACKET_DATA | UINT8[x] | If PACKET_LEN was specified, this contains the array of data that should be sent in the NAV acoustic packet, where “x” is the value specified in PACKET_LEN.<br><br>If PACKET_LEN was not sent, this data should not be sent either. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK — The packed has been queued.<br>• CST_CMD_PARAM_INVALID — The value of one of the parameters is invalid.<br>• CST_CMD_PARAM_MISSING — There are not enough data parameters provided to satisfy the message requirements. |
| DEST_ID | BID_E | The beacon ID code of the queue that has been modified/queried.<br>Valid values are from 0 to 100. |
| PACKET_LEN | UINT8 | The length of the packet in the queue for the specified DEST_ID beacon.<br>If DEST_ID was BEACON_ALL (0), then this will return the new packet length loaded, unless PACKET_LET was omitted then 0 is returned. |

### 8.4.6. CID_NAV_QUEUE_CLR

The CID_NAV_QUEUE_CLR command is used to clear any packed that has been queued for remote retrieval by the NAV protocol, using the CID_NAV_QUEUE_SET command.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_CLR) |
| DEST_ID | BID_E | Optional ID code of the beacon to that the queued packet will be sent to in response to a CID_NAV_QUEUE_SEND command. Requests received from other beacons will be ignored. Valid values are from 0 to 100.<br><br>A value of BEACON_ALL (0) indicates data is queued for all destinations. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_CLR) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK — The NAV command is being sent. |
| DEST_ID | BID_E | The beacon ID code of the queue that has been cleared. Valid values are from 0 to 100. |

### 8.4.7. CID_NAV_QUEUE_STATUS

The CID_NAV_QUEUE_STATUS command is used to determine if any packet is queued and waiting for remote retrieval by the NAV protocol in response to a CID_NAV_QUERY_SEND command.

The number of bytes in the packet payload is returned in response to this command, and when the packet has been delivered, this value is reported as 0, indicating empty (and another packet can be queued).

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_STATUS) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_STATUS) |
| PACKET_LEN | UINT8[16] | Array of 16 lengths representing the pending number of data bytes queued for each beacon.<br>Valid values for each entry are from 0 to **30**, except for PACKET_LEN[0] which will always return 0.<br>A value of 0 indicates there is no queued data. |

### 8.4.8. CID_NAV_STATUS_SEND

The CID_NAV_STATUS_SEND message is used to broadcast an application defined status data packet to all other beacons in the network (using a OWAY message type).

The payload includes an additional BEACON_ID field allowing the contents to be specified as applying to a specific beacon.

Although the remaining data content of the message is left up to the developer, it is anticipated that this type of message could be used to broadcast the resolved position (i.e. Depth, Latitude/Longitude or Northing/Easting) of a specific beacon after a PING or NAV_QUERY event has been performed by the controlling USBL beacon in a 'round-robin' scheme.
Some examples of data payload content are discussed at the end of this section.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_STATUS_SEND) |
| BEACON_ID | BID_E | The ID code of the beacon that this status message is referring to (not the receiving beacon, as status messages are broadcast to everyone). Valid values are from 1 to 100, and 0 for BEACON_ALL.<br><br>NB: This ID code does not necessarily have to refer to the local sending beacon (although it may). |
| PACKET_LEN | UINT8 | Optional number of bytes to send in the NAV acoustic packet. Valid values are from 0 to **30** (not 31!).<br><br>Omit this parameter (and PACKET_DATA) or use a value of 0 if not required. |
| PACKET_DATA | UINT8[x] | If PACKET_LEN is specified greater than 0, then these fields specify the array of data that should be sent in the NAV acoustic packet, where "x" is the value specified in PACKET_LEN. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_STATUS_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully. |

| | | |
|---|---|---|
| | | Valid values are…<br>• CST_OK — The PING command is being sent.<br>• CST_CMD_PARAM_INVALID — The DEST_ID parameter is invalid.<br>• CST_CMD_PARAM_MISSING — The DEST_ID has not been specified correctly.<br>• CST_XCVR_BUSY — The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| BEACON_ID | BID_E | The ID code of the beacon that the status data is referring to. Valid values are from 0 to 100. |

**Examples of PACKET_DATA content**

The specific formatting of the PACKET_DATA field is let up to the developer for their application. However, the examples below show how certain data types can be efficiently coded into the payload…

| | | |
|---|---|---|
| *POSITION_DEPTH* | *INT16* | *The vertical Depth distance component of the remote beacon from the surface.*<br>*This value is encoded in deci-meters, so multiply the meters distance by 10 and truncate as an integer.* |
| *POSITION_LATITUDE*<br>*POSITION_LONGITUDE* | *INT32* | *The latitude/longitude part of a resolved beacons position, or GPS coordinate.*<br>*This value is encoded in milliseconds of arc angle, so divide by 3600000 to obtain a value in decimal degrees.*<br>*This is more efficient that sending the position as a DOUBLE (8-byte) floating point number.* |
| *POSITION_EASTING*<br>*POSITION_NORTHING* | *INT16 or INT32* | *The Easting/Northing distance component of the relative or absolute position of the remote beacon to the local beacon computed from the range, incoming signal angle, local beacon depth, attitude and magnetic heading.*<br>*This value must be encoded in deci-meters, so multiply the meters distance by 10 and truncate as an integer.* |

### 8.4.9. CID_NAV_STATUS_RECEIVE

The CID_NAV_STATUS_RECEIVE message is generated when the beacon receives acoustic information sent from another beacon with the CID_NAV_STATUS_SEND command.
On reception of the message, the acoustic payload is output from the serial port.

NAV_STATUS messages are broadcast from the sender to all other beacons with a OWAY type message, but the payload includes an additional BEACON_ID field allowing the contents to be specified as applying to a specific beacon.
Although the packet data content of the message is left up to the developer, it is anticipated that this type of message could be used to broadcast the resolved position (i.e. Latitude and Longitude) of a specific beacon after a PING or NAV_QUERY event has been performed by the controlling USBL beacon in a 'round-robin' scheme.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_STATUS_RECEIVE) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to signal from the beacon sending data.<br>This will not contain range or position information, but will contain the message SRC_ID. |
| BEACON_ID | BID_E | The ID code of the beacon that this status message is referring to (not the receiving beacon, as status messages are broadcast to everyone). Valid values are from 1 to 100, and 0 for BEACON_ALL.<br><br>NB: This ID code does not necessarily refer to the local sending beacon (although it may). |
| PACKET_LEN | UINT8 | The number of bytes sent in the NAV acoustic packet. Valid values are from 0 to **30** (not 31!). |
| PACKET_DATA | UINT8[x] | The array of data received in the NAV acoustic packet, where "x" is the value specified in PACKET_LEN. |
| LOCAL_FLAG | BOOLEAN | True if the BeaconID in the status message refers to this local beacon.<br>This flag is the same as the Boolean result of logical test "BEACON_ID = LOCAL_BEACON_ID" |

### 8.4.10. CID_NAV_QUEUE_GET

The CID_NAV_QUEUE_GET command is used to retrieve the packet contents that has been queued up to send to a remote beacon.

The number of bytes in the packet payload, as well as the payload itself is returned in response to this command, a value of zero means that there was no queued data for the beacon.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_GET) |
| DEST_ID | BID_E | The ID code of the beacon whose queued packet is being retrieved.<br>Valid values are from 1 to 100.<br><br>Data cannot be retrieved for the same ID as that used by the beacon itself. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_NAV_QUEUE_GET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The queued packet has been retrieved. |
| • CST_CMD_PARAM_INVALID | | The value of one of the parameters is invalid. |
| • CST_CMD_PARAM_MISSING | | There are not enough data parameters provided to satisfy the message requirements. |
| • DEST_ID | | The beacon ID code of the queue that has been retrieved.<br>Valid values are from 1 to 100. |
| PACKET_LEN | UINT8 | The number of bytes in the queued packet. |
| PACKET_DATA | UINT8[x] | The array of data that has been queued, where “x” is the value specified in PACKET_LEN. |

## 8.5. CFG Protocol Messages

The Acoustic Configuration protocol has been developed to allow the remote management of beacon operating parameters by sending and receiving acoustic messages rather than relying on direct connection of the beacon to a PC for configuration.

This is particularly useful for querying and setting the Beacon ID codes for each modem beacon to be tracked in a positioning system from a single X150 head (or similar).
In addition to Beacon ID, other critical parameters to correct operation (such as Range Timeout or Response Turnaround) call also be specified and queried.

Addressing of remote beacons is done by their unique 16-bit serial number (specified at manufacturing time) rather than by the 4-bit Beacon ID which can easily be set to duplicate values and prevent operations.

Two commands are provided (CID_CFG_BEACON_GET and CID_CFG_BEACON_SET) which are used to query or adjust values respectively.
In response to either command, at some point later a received response will cause the beacon to output a CID_CFG_BEACON_RESP message on its serial port containing the values being used by the remote beacon.

The acoustic interrogation mechanism uses one-way broadcast messages (OWAY), not the request/response mechanism used for other communications (such as PING, ECHO and DAT protocols), so ranging cannot be performed from the returned response, and no timeout errors are generated should a response not be received. The user should manually ensure the appropriate length of time is observed for a response to be received before starting another transmission.

This protocol can be locked/unlocked using the CFG V2 protocol.

### 8.5.1. CID_CFG_BEACON_GET

The CID_CFG_BEACON_GET command is issued to query the Acoustic Transceiver settings of a remote beacon, addressing it by its unique 16-bit serial number rather than Beacon ID.

At some point later after this message is sent and locally acknowledged by the beacon, a further CID_CFG_BEACON_RESP message will be output from the remote beacon receiving the acoustic command.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_BEACON_GET) |
| DEST_SERIALNUMBER | UINT16 | The 16-bit encoded value of serial number for the Beacon that should process & response to this message. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_BEACON_GET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The CFG command is being sent.<br>• CST_CMD_PARAM_MISSING – The is not enough data parameters provided to satisfy the message requirements.<br>• CST_XCVR_BUSY – The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| DEST_SERIALNUMBER | UINT16 | The 16-bit encoded value of serial number for the Beacon that the message was sent to. |

### 8.5.2. CID_CFG_BEACON_SET

The CID_CFG_BEACON_SET command is issued to specify and apply new Acoustic Transceiver settings to a remote beacon, addressing it by its unique 16-bit serial number rather than Beacon ID.
When the remote beacon receives the acoustic command, it will change and store the new settings in permanent Flash memory.

At some point later after this message is sent and locally acknowledged by the beacon, a further CID_CFG_BEACON_RESP message may be output if the remote beacon receives and applies the new settings.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_BEACON_SET) |
| DEST_SERIALNUMBER | UINT16 | The 16-bit encoded value of serial number for the Beacon that should process & response to this message. |
| NEW_BEACON_ID | BID_E | The new ID code the beacon should response and reply to in future acoustic communications.<br>Valid values are from 1 to 100.<br>Specifying a value of 0 can be used to ensure the current setting is not changed. |
| NEW_RANGE_TMO | UINT16 | The new value of Range Timeout to use, specified in meters.<br>Specifying a value of 0 can be used to ensure the current setting is not changed. |
| NEW_RESP_TURNAROUND | UINT16 | The new value of Response Turnaround Time to use, specified in milliseconds (all beacons MUST use the same value to prevent error in ranging from being observed).<br>Specifying a value of 0 can be used to ensure the current setting is not changed. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_BEACON_SET) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The CFG command is being sent. |

| | | |
|---|---|---|
| | | • CST_CMD_PARAM_MISSING – The is not enough data parameters provided to satisfy the message requirements.<br>• CST_XCVR_BUSY – The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| DEST_SERIALNUMBER | UINT16 | The 16-bit encoded value of serial number for the Beacon that the message was sent to. |

### 8.5.3. CID_CFG_BEACON_RESP

The CID_CFG_BEACON_RESP message is sent from the serial port of the beacon when it received the acoustic message back from a remote beacon that had been initiated as the result of a CID_CFG_BEACON_GET or CID_CFG_BEACON_SET command.
The message contains the Acoustic Transceiver settings in use by the queries remote beacon.

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_BEACON_RESP) |
| SRC_SERIALNUMBER | UINT16 | The 16-bit encoded value of serial number for the Beacon that was queried. |
| BEACON_ID | BID_E | The Beacon ID code in use by the remote beacon.<br>Valid values are from 1 to 100. |
| RANGE_TMO | UINT16 | The value of Range Timeout in use, specified in meters. |
| RESP_TURNAROUND | UINT16 | The value of Response Turnaround Time in use, specified in milliseconds (all beacons MUST use the same value to prevent error in ranging from being observed). |
| SUPPORTS_EXT_ADDR | BOOL | Indicates that the remote beacon can accept a beacon ID up to 100 (i.e. the beacon is on >=FW3.3). |
| SUPPORTS_CFG_V2 | BOOL | Indicates that the remote beacon supports the CFG Protocol V2 Messages. |

## 8.6. CFG Protocol Version 2 Messages

The Acoustic Configuration protocol version 2 has been developed to allow control and querying of more beacon parameters and functions, in a more flexible way than the original CFG protocol (note that the original protocol has not been removed). The new protocol also introduces the ability to send configuration messages intended for a device connected to the beacon's serial port rather than for the beacon itself.

Addressing of remote beacons is still done by their unique serial number (specified at manufacturing time) rather than by the 4-bit Beacon ID which can easily be set to duplicate values and prevent operations.

When configuring or querying values internal to the beacon itself, configuration parameters are queried or set by their "address" (see CFG_ADDRESS_E). Multiple addresses can be queried and/or set in the same command. When a "set" address is used, if the set was unsuccessful the response will contain the "set" address along with the current value of that parameter, otherwise (on successful set) the response will not contain the "set" address. Any parameters that have been set will NOT persist over a power cycle.

Two commands are provided; CID_CFG_REQ_SEND and CID_CFG_RESP_SEND. The first is used to set and get values on the remote beacon. The second is for use in cases where the configuration sent with CID_CFG_REQ_SEND is intended for the device connected to the beacon's serial port, in which case that device must initiate the response using the CFG_RESP_SEND command.
In response to the CID_CFG_REQ_SEND command, at some point later a received response will cause the beacon to output a CID_CFG_RESP_RECEIVE message on its serial port containing the requested values from the remote beacon.

The acoustic interrogation mechanism uses one-way broadcast messages (OWAY), not the request/response mechanism used for other communications (such as PING, ECHO and DAT protocols), so ranging cannot be performed from the returned response, and no timeout errors are generated should a response not be received. The user should manually ensure the appropriate length of time is observed for a response to be received before starting another transmission.

The CFG protocol V2 can be locked (acoustically or serially) with a 16-bit PIN, preventing any further acoustic configuration changes to be made until the protocol is acoustically unlocked with the same 16-bit PIN, or serially unlocked. This locking mechanism is activated through the use of a CID_CFG_REQ_SEND containing the CFG_ADDRESS_CFG_LOCK or CFG_ADDRESS_CFG_UNLOCK addresses to perform the relevant action acoustically or using the CID_CFG_LOCAL_LOCK and CID_CFG_LOCAL_UNLOCK commands serially. N.B. This will also lock/unlock version 1 of the protocol.

### 8.6.1. CID_CFG_REQ_SEND

The CID_CFG_REQ_SEND command is issued to set or query parameters of a remote beacon, addressing it by its unique 16-bit serial number rather than Beacon ID.

At some point later, a further CID_CFG_RESP_RECEIVE message will be output from the beacon if and when it receives an acoustic response.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_REQ_SEND) |
| DEST_SERIALNUMBER | UINT32 | The 32-bit encoded value of serial number for the Beacon that should process & respond to this message. |
| TARGET | UINT8 | The target for the configuration, possible values are:<br>• 0 – The remote beacon itself<br>• 1 – The main serial port of the remote beacon |
| If Target = 0 then one or more address or address/value pair as follows: | | |
| CFG_ADDRESS | CFG_ADDRESS_E or UINT8 | The address of a config item to get or set. |
| CFG_VALUE | ANY | If CFG_ADDRESS is a “set” address, the value to set for the parameter at that address. |
| Else if Target = 1 then: | | |
| CFG_LEN | UINT8 | The length of the config data packet. |
| CFG_DATA | UINT8[x] | The array of config data that should be sent in the CFG acoustic packet, where “x” is the value specified in CFG_LEN. The format of this data packet is application defined. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_REQ_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The CFG command is being sent.<br>• CST_CMD_PARAM_MISSING – There are not enough data parameters provided to satisfy the message requirements. |

| | | |
|---|---|---|
| | | • CST_XCVR_BUSY – The message cannot be sent as the acoustic transceiver is busy performing another operation. |
| DEST_SERIALNUMBER | UINT32 | The 32-bit encoded value of serial number for the Beacon that the message was sent to. |

### 8.6.2. CID_CFG_REQ_RECEIVE

The CID_CFG_REQ_RECEIVE message is sent from the serial port of the beacon when it receives an acoustic config request from another beacon (sent using the CID_CFG_REQ_SEND message), and the TARGET parameter is not 0. If the TARGET parameter is set to 1, this message will be sent out of the main UART (the serial port on the external connector).

If this message is received and the IS_BROADCAST field is false, the device connected to the SeaTrac serial port is required to respond using the CID_CFG_RESP_SEND command within a reasonable time frame (application defined).

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_REQ_RECEIVE) |
| IS_BROADCAST | BOOLEAN | True if the received config was addressed to all beacons. |
| CFG_LEN | UINT8 | The length of the config data packet received. |
| CFG_DATA | UINT8[x] | The array of config data received in the CFG acoustic packet, where “x” is the value specified in CFG_LEN. The format of this data is application defined. |

### 8.6.3. CID_CFG_RESP_SEND

The CID_CFG_RESP_SEND command is issued to send a response having received an acoustic config request from a remote beacon with the TARGET parameter set to a non-zero value.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_RESP_SEND) |
| CFG_LEN | UINT8 | The length of the config response data packet. |
| CFG_DATA | UINT8[x] | The array of config response data that should be sent in the CFG acoustic packet, where "x" is the value specified in CFG_LEN. The format of this data is application defined. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_REQ_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The CFG command is being sent.<br>• CST_CMD_PARAM_MISSING – There are not enough data parameters provided to satisfy the message requirements.<br>• CST_XCVR_BUSY – The message cannot be sent as the acoustic transceiver is busy performing another operation. |

### 8.6.4. CID_CFG_RESP_RECEIVE

The CID_CFG_RESP_RECEIVE message is sent from the serial port of the beacon when it receives an acoustic message back from a remote beacon that has been initiated as the result of a CID_CFG_REQ_SEND command.

The message contains the values of requested parameters

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG _RESP_RECEIVE) |
| SRC_SERIALNUMBER | UINT32 | The 16-bit encoded value of serial number for the Beacon that was queried. |
| TARGET | UINT8 | The target for the configuration, possible values are:<br>• 0 – The remote beacon itself<br>• 1 – The main serial port of the remote beacon |
| If Target = 0 then zero or more address/value pair as follows: | | |
| CFG_ADDRESS | CFG_ADDRESS_E or UINT8 | The address of a config item. |
| CFG_VALUE | ANY | If CFG_ADDRESS is a “set” address this indicates that the set failed, this field will contain the current value of the parameter at that address. If CFG_ADDRESS is a “get” address, the value of the config parameter at that address. |
| Else if Target = 1 then: | | |
| CFG_LEN | UINT8 | The length of the config response data packet. |
| CFG_DATA | UINT8[x] | The array of config response data that was received in the CFG acoustic packet, where “x” is the value specified in CFG_LEN. The format of this data packet is application defined. |

### 8.6.5. CID_CFG_LOCAL_LOCK

The CID_CFG_LOCAL_LOCK command is issued to lock the receiving beacon's acoustic configuration protocol, preventing it from acting on or responding to acoustic configuration requests.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_LOCAL_LOCK) |
| LOCK_CODE | UINT16 | The code to lock the beacon with. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_LOCAL_LOCK) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The beacon has been locked.<br>• CST_CMD_PARAM_MISSING – There are not enough data parameters provided to satisfy the message requirements.<br>• CST_CFG_ALREADY_LOCKED – The beacon is already locked. |
| LOCK_CODE | UINT16 | The code that the beacon is locked with. This may be the code supplied in the command message, or if the STATUS field contains the CST_CFG_ALREADY_LOCKED value, it will be the code that was used to lock the beacon originally. |

### 8.6.6. CID_CFG_LOCAL_UNLOCK

The CID_CFG_LOCAL_UNLOCK command is issued to unlock the receiving beacon's acoustic configuration protocol, allowing it to act on and respond to acoustic configuration requests.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_LOCAL_UNLOCK) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_CFG_LOCAL_UNLOCK) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are…<br>• CST_OK – The beacon has been unlocked.<br>• CST_CFG_NOT_LOCKED – The beacon is not locked, the command has had no effect. |

## 8.7. TRACK Protocol Messages

The TRACK protocol has been developed to provide a way of tracking a beacon more quickly when no data transmission is required and is designed to be used to track a single remote beacon (although it can be used for multiple if desired).

The TRACK protocol is a little different to the other protocols in that a single request can trigger multiple responses from the remote beacon. The request includes the number of responses desired from the remote beacon, and positions are then calculated for each response. Up to 16 responses can be requested from a single request allowing a faster update rate to be achieved. Each response contains a sequence number so the local beacon can correct for any responses it doesn't hear, and the surface can also see when a response is not heard.

The TRACK protocol is restricted to being used with beacon IDs below 16. This helps to keep the protocol as fast as possible.

### 8.7.1. CID_TRACK_SEND

The CID_TRACK_SEND command is issued to initiate a TRACK protocol exchange between the local beacon and a remote beacon, from which its range and position can be determined up to 16 times.

For positioning, acoustic TRACK messages are designed to provide the fastest update rate for tracking a single remote beacon. The TRACK protocol can be used for multiple remote beacons, but each TRACK exchange must finish before the next can begin, so if 16 responses are requested from a remote beacon, all 16 (or errors) must be received before sending a request to the next beacon.



If successful, the CID_TRACK_SEND command will start an acoustic message transmission, and at some time later an acoustic response will be heard. This may be followed by more acoustic responses if requested in the initial transmission.
The TRACK protocol handler will generate subsequent serial message to signal when these events occur. Additionally, acoustic transceiver diagnostic messages are available to monitor activity and obtain further information on the received signal levels – see section 7.6 on page 108 for further details.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_TRACK_SEND) |
| DEST_ID | BID_E | The ID code of the beacon to send the acoustic message to, and receive position and ranging information for.<br>Valid values are from 1 to 15. |
| RESPONSE_COUNT | UINT8 | Value specifying the number of responses to request from the remote beacon. Valid values are from 1 to 16. |

### 8.7.2. CID_TRACK_REQ

The CID_TRACK_REQ status message is generated when the beacon received an acoustic TRACK message from the sending beacon. This message is provided by the protocol-handler in the beacon for information purposes only, and no action or acknowledgment by the external system/user is required.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_TRACK_REQ) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to signal from the beacon sending data.<br>This will not contain range or position information. |
| RESPONSE_COUNT | UINT8 | Value specifying the number of responses requested by the sending beacon. |

### 8.7.3. CID_TRACK_RESP

The CID_TRACK_RESP status message is generated when the beacon received an acoustic TRACK message from the sending beacon. This message is provided by the protocol-handler in the beacon for information purposes only, and no action or acknowledgment by the external system/user is required.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_TRACK_RESP) |
| ACO_FIX | ACOFIX_T | A Fix structure containing information relating to the range and position of the beacon sending data. |
| RESPONSE_NO | UINT8 | Value specifying the sequence number of this response. |

### 8.7.4. CID_TRACK_ERROR

The CID_TRACK_ERROR status message is generated when a TRACK operation is not successful, usually because a response is not received from the remote interrogated beacon, causing a timeout.

This message will only be generated with a timeout status if the first response from the remote beacon is not heard, not when any subsequent expected responses are not heard. Note that it is possible for the local beacon to miss the first response from the remote beacon, and therefore generate this message, while the remote beacon is still sending responses which may then also be reported by the local beacon.



This message is a status message that may be sent by the beacon at any time (not in response to a command message) depending on acoustic activity triggering a transceiver event.

**Status Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_TRACK_ERROR) |
| STATUS | CST_E | Status code indicating the error that occurred.<br>Valid values are… |
| • CST_XCVR_RESP_TIMEOUT | | A valid response message was not received in the time allowed by the beacons Range-Timeout setting, indicating the remote beacon cannot be heard, is not present or lies outside this range. |
| • CST_XCVR_RESP_WRONG | | A message from another beacon was received while waiting for a response. |
| • CST_XCVR_RESP_ERROR | | A general reception error has occurred while waiting for a response. |
| BEACON_ID | BID_E | The ID code of the beacon that the error applies to (i.e. the DEST_ID value used in the CID_TRACK_SEND command).<br>Valid values are from 1 to 16. |

## 8.8. PINGER Protocol Messages

The PINGER Protocol controls the generation of 10ms continuous wave 37kHz pulses as used by locator beacons.

### 8.8.1. CID_PINGER_SEND

The CID_PINGER_SEND command is issued to cause the local beacon to produce a single 10ms continuous wave 37kHz pulse.



If successful, the CID_PINGER_SEND command will start an acoustic pulse transmission.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PINGER_SEND) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PINGER_SEND) |
| STATUS | CST_E | Status code indicating whether the command succeeded. Valid values are… |
| • CST_OK | | Command was successful. |
| • CST_XCVR_BUSY | | The pulse could not be transmitted as the XCVR module was busy. |

### 8.9.1. CID_PINGER_SET_ENABLED

The CID_PINGER_SET_ENABLED command is issued to cause the local beacon to enable or disable pinger locator mode. In this mode, a single 10ms continuous wave 37kHz pulse will be transmitted every second.



If successful, the CID_PINGER_SET command will start a repeated acoustic transmission.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PINGER_SEND) |
| ENABLE | BOOLEAN | Whether to enable or disable pinger locator mode. |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_PINGER_SEND) |
| STATUS | CST_E | Status code indicating whether the command succeeded. Valid values are… |
| • CST_OK | | Command was successful. |
| • CST_PARAM_MISSING | | The command was missing the ENABLE parameter. |

## 8.11. INTERRUPT Protocol Message

The INTERRUPT protocol commands control the generation of an interrupt tone that can be used halt a system. When an interrupt tone is received by a beacon it will output a CID_INTERUPT_RECEIVE message and temporarily disable its transceiver. Then, sometime later it will reset the transceiver to an idle state and output a CID_INTERRUPT_RESET_IDLE message, whilst keeping the interrupt send mechanism locked for a longer period of time. At this point, any communication that may have been happening between beacons in the system will have ceased and all beacons will be in an idle state waiting for a command.

### 8.11.1. CID_INTERRUPT_SEND

The CID_INTERRUPT_SEND command will cause the beacon to generate an interrupt tone.



If successful, the CID_INTERRUPT_SEND command will start an acoustic transmission.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_INTERRUPT_SEND) |

**Response Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_INTERRUPT_SEND) |
| STATUS | CST_E | Status code used to indicate if the command executed successfully.<br>Valid values are… |
| • CST_OK | | The INTERRUPT tone is being sent. |
| • CST_INTERRUPT_LOCKED | | The INTERRUPT tone cannot be sent as the protocol |

### 8.11.2. CID_INTERRUPT_RECEIVE

The CID_INTERRUPT_RECEIVE message indicates that the beacon has received an interrupt tone from another beacon in the system. When this message is received from a beacon, its transceiver will be in a disabled state for a period of time, the end of which will be indicated with a CID_INTERRUPT_RESET_IDLE message. It will also be unable to transmit its own interrupt tone for a further period of time; the end of this period will be indicated with a CID_INTERRUPT_UNLOCK message.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_INTERRUPT_RECEIVE) |

### 8.11.3. CID_INTERRUPT_RESET_IDLE

The CID_INTERRUPT_RESET_IDLE message indicates that the beacon is now able to receive and transmit following a period of locked transceiver due to sending or receiving an interrupt tone.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_INTERRUPT_RESET_IDLE) |

### 8.11.4. CID_INTERRUPT_UNLOCK

The CID_INTERRUPT_UNLOCK message indicates that the beacon is now allowed to transmit its own interrupt tone if required, following a period of this not being allowed due to receiving an interrupt tone.

**Command Message Parameters**

| *Parameter* | *Type* | *Description* |
|---|---|---|
| MSG_ID | CID_E | Command identification code (CID_INTERRUPT_UNLOCK) |

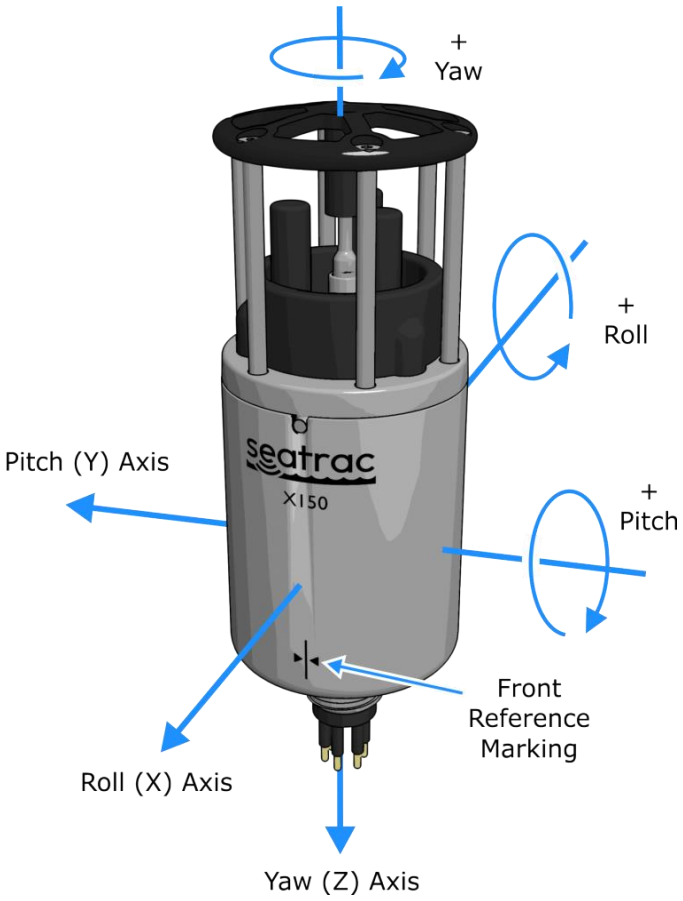
# 9. Beacon Definitions and Frames of Reference

The diagrams below show the definitions of the SeaTrac Beacon's frames-of-reference…

## 9.1. Attitudes (Yaw, Pitch and Roll)

For rotation and orientation, the beacon uses the following axis definitions.
These follow the same standard as those used to define aircraft attitudes and rotations, allowing the "right-hand-rule" to be applied to each axis.

By orienting the positive Z-axis downwards, positive Yaw angles also match compass headings, with 0° Yaw occurring when the housing "front" marking aligns with magnetic north.
Pitch and Roll angles are 0° when gravity is perpendicular to (and below) the horizontal plane, defined by the XY axis pair.

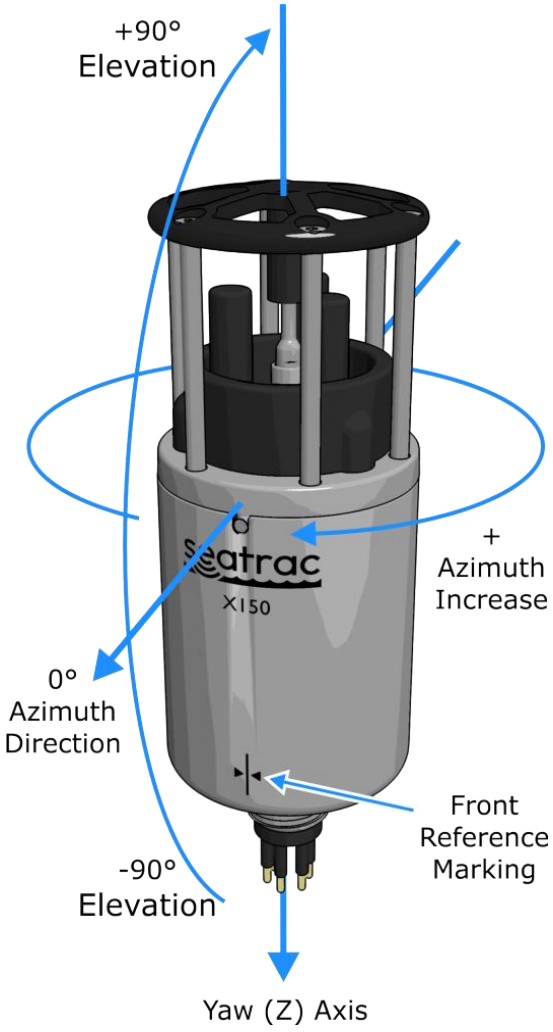



Attitudes are valid where…

- 0° ≤ Yaw < 360°
- -90° ≤ Pitch ≤ +90°
- -180° ≤ Roll < +180°

## 9.2. USBL Spherical Angles (Azimuth and Elevation)

On X150 beacons, for resolving incoming USBL signals, positive azimuths are defined as being a positive rotation around the yaw axis (complying with the right-hand-rule). Positive elevations are defined as being above the beacons horizontal plane (defined by the XY axis pair), while negative elevations are defined as being below it.

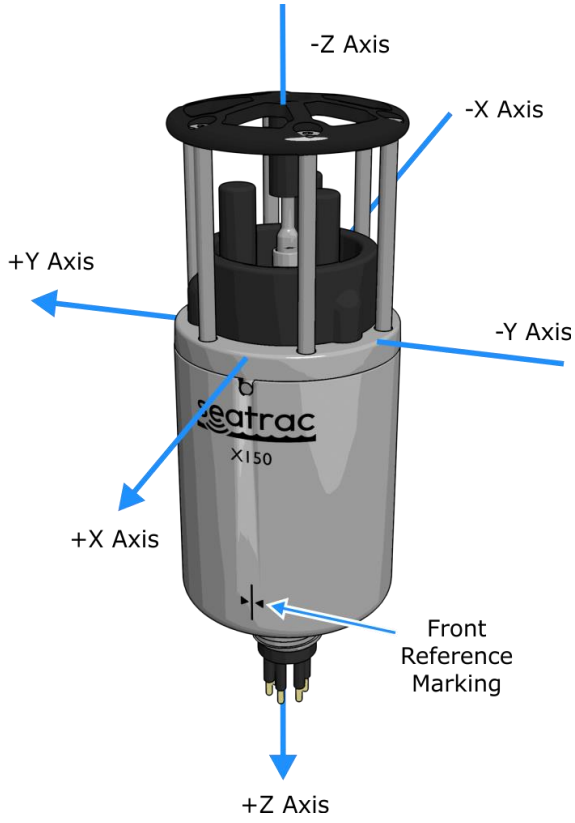



Spherical angles are valid where…

- $0° \leq \text{Azimuth} < 360°$
- $-90° \leq \text{Elevation} \leq +90°$

## 9.3. USBL Local Relative Position Coordinates

On X150 beacons, having resolved the range and incoming USBL signal angle (as an azimuth and elevation), the position of the remote beacon in the beacon's local frame of reference is computed (in metres) and defined in the coordinate frame shown below…




Coordinates are relative to the beacon's position, with the origin being the defined as the centre of the upper beacon housing bulkhead.

Please note that to comply to the earlier 'attitude' definition, positive Z axis directions are defined as "down" (towards the seabed) when the beacon is mounted in the upright position shown above.

## 9.4. USBL World Relative Position Coordinates

On X150 beacons, once a local frame-of-reference position is computed, it is transformed into the world frame-of-reference using the attitude from the AHRS (or user specified setting).

World coordinates are provided in a Cartesian Northing, Easting and Depth (NED) triplet, with values stated in metres. Positive values of depth represent a translation towards the seabed, while negative values represent a translation towards the surface.



Distributor…

www.blueprintsubsea.com

**Blueprint Subsea,**
The Clock Tower Business Centre,
Low Wood, Ulverston, Cumbria, LA12 8LY, UK