3D at Depth

3D at Depth, Inc

SL Sensor

Interface Control Document

Version C9

November, 2021

Change Record

Version	Date	Notes	Author
0.1	Oct-2017	Initial Release	B. Nickerson
1.1	Nov-2017	Updates per ICD meeting	B. Nickerson
C.1	Feb-2019	New Release to Customer	J. Jeong
C.2	Mar-2019	Updated after Customer mtg.	J. Jeong
A.3	Apr-2019	Structure Updated	J. Jeong
C.4	Jul-2019	FaultCode Structure Updated, Changed checksum calculation	J. Jeong
C.5	Aug-2019	Fix Status packet definition, OPBTemp_C and RTV Config OpenTCPServer. Changes highlighted	D. Florin
C.6	Sep-2019	Response of RTV config is not “ACK” but a range limited CFG packet	D. Florin
C.7	Mar-2020	Changed the fault order and reused PitchRollLogRate_s for Version of the structure, no changes to the structure sizes.	D. Florin
C.8	Nov-2020	Added definition of the Version 1 CCI interface; Adds Pan/Tilt, wait and a scripting mode.	D. Florin
C.9	Nov-2021	Updates – 3D PanTilt, Gain start/end, Some SL4/5.	D. Florin

Table of Contents

1 Introduction

The primary objective of this Interface Control Document (ICD) is to document the standards for Ethernet communication for command and control of the SL sensor and the transfer of 3D data measurements for a low bandwidth interface to a SL type system.

2 Scope

This ICD describes the TCP/IP and UDP software interfaces to the 3D at Depth Subsea Lidar sensor (SL). It also details the necessary communication and handshaking protocols between the sensor and the commanding device. This is considered a working document until the protocol has been defined.

Figure 1 below is a schematic of the overall communication plan.

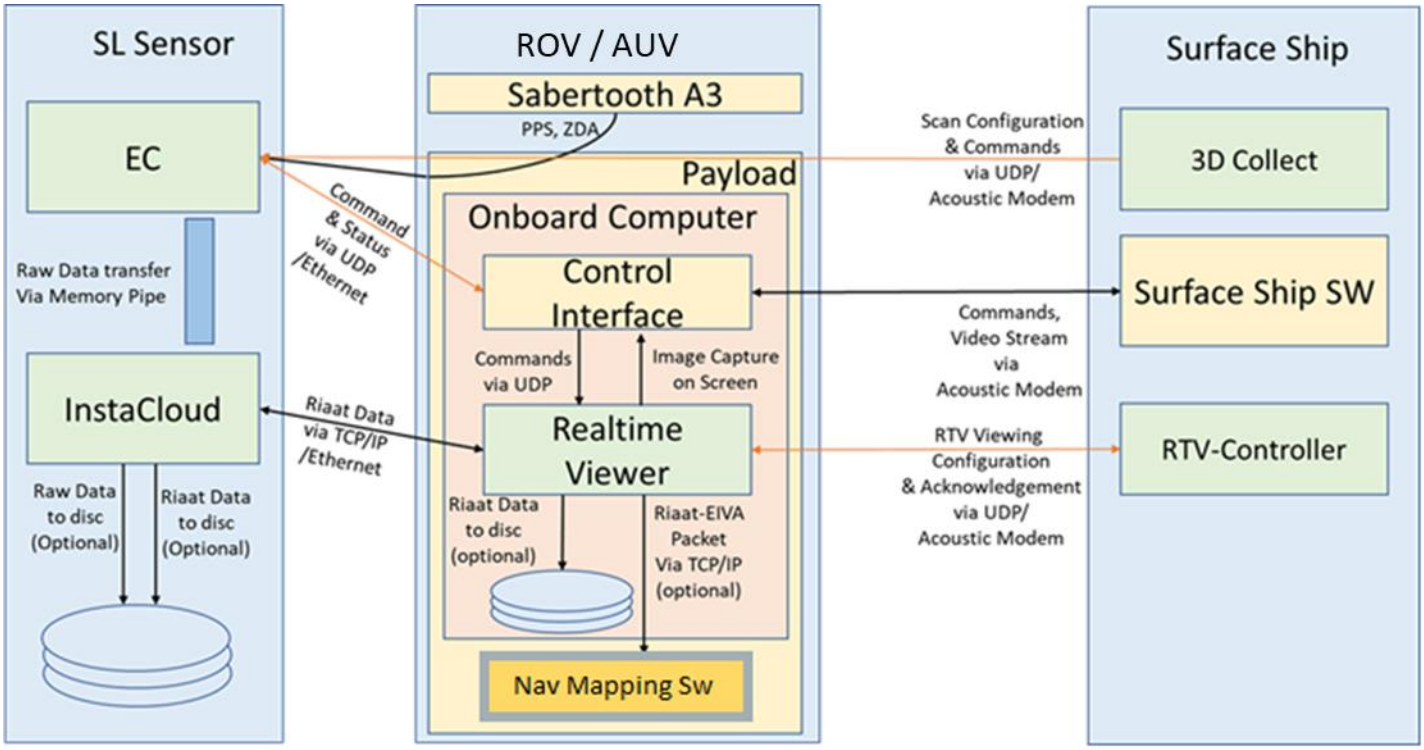
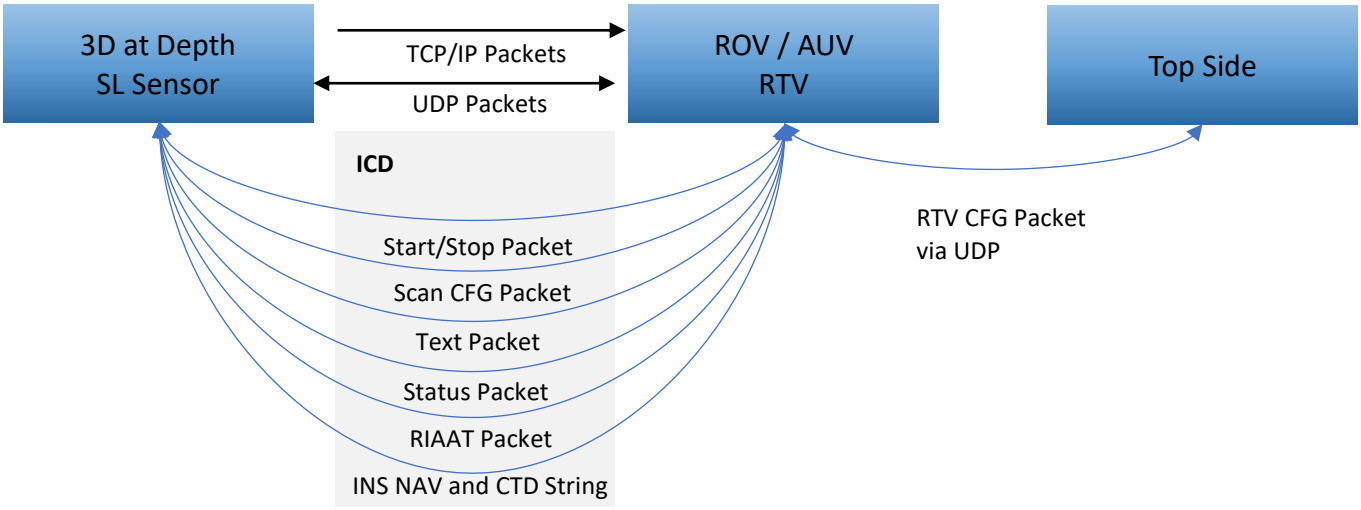
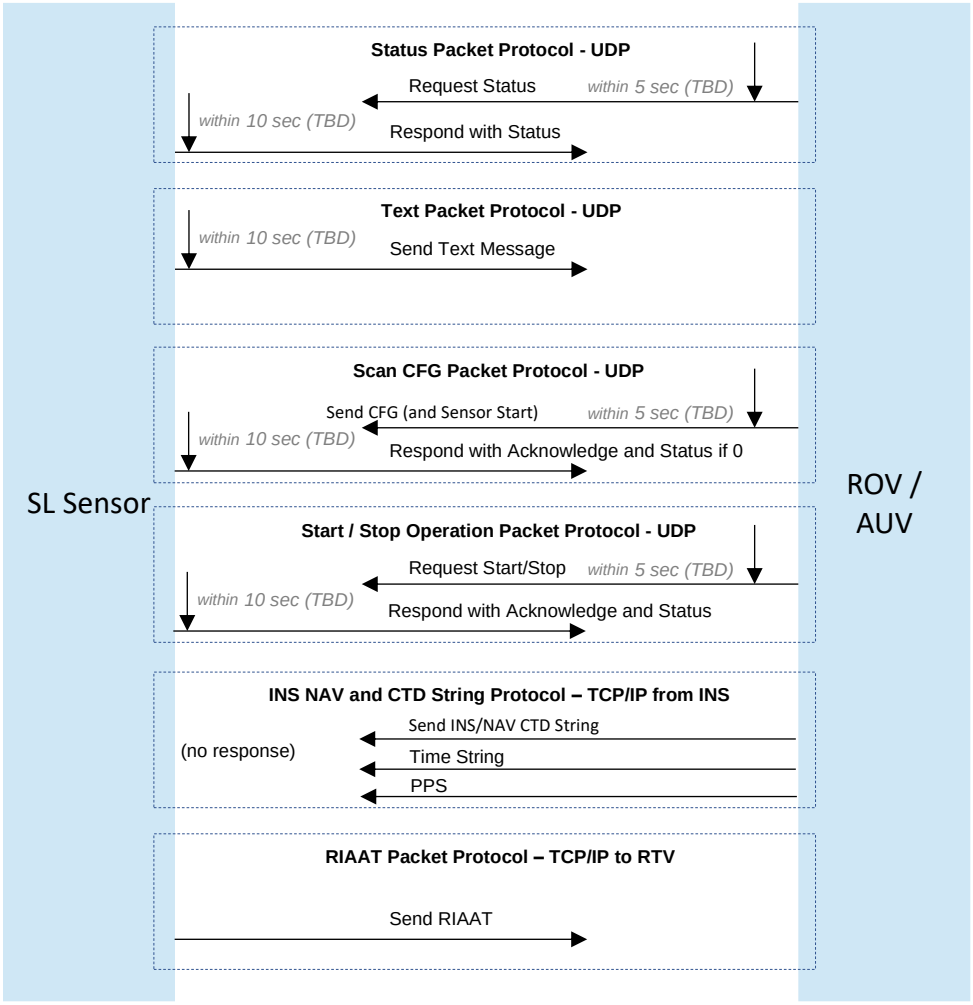


Figure 1. Overall Communication Schematic

3 Ethernet Interface

Communication packets between the SL Sensor and the host are sent and received via the TCP/IP and UDP Ethernet Protocol. The SL sensor is configured with a static IP address (although we can be configured for dynamic operations) for its network connections. The specific send-receive, handshake acknowledgment standard is shown below with the per packet usage. The corresponding handshake mechanism is also represented in the figure.





Number Representations

The byte order for multi-byte packet fields will comply with the little-endian format (Intel). The table below describes the number of bytes per types. All structures and typedefs must adhere to a data pack, single byte scheme, as some packets may align to 1 byte size.

Type	Number of bytes
char, unsigned or signed 8 bit integer	1
short, unsigned or signed 16 bit integer	2
long, unsigned or signed 32 bit integer	4
float, 4 byte IEEE floating point representation	4
double, 8 byte IEEE floating point representation	8

Table for C++ definitions of the various types.

C++ type	Definition	Number of bytes
INT8	signed char	1
INT16	signed short	2
INT32	signed int	4
UINT8	unsigned char	1
UINT16	unsigned short	2
UINT32	unsigned int	4
INT64	signed __int64	8
UINT64	unsigned __int64	8

Packet Header

All packets transferred between the SL Sensor and the host consist of a packet header and packet data. The general packet header is described below. In particular, the application should read the fixed size header, then check the Alignment Code, and then the Packet ID to determine what type of data follows. The Data Size element can then be utilized to read all subsequent data in the packet. Version is used to determine the version of the response. Version 0 is only for scanning and does not support Pan Tilt, wait or scripting, version 1 adds in the pan tilt, timed wait and scripting.

Packet Header		
Item	**Value**	**Bytes**
Alignment Project Code	0x3DF0	2 (1 UINT16)
Packet ID	0X0Dnn	2 (1 UINT16)
Version	0 or 1	2 (1 UINT16)
Data Size (bytes)	varies	4 (1 UINT32)

Checksum

At the end of each packet is a checksum. The Checksum is a UINT16 sum of the whole packet data excluding the Checksum value itself (set to zero). The routine for calculating CheckSum is below, where `m_pBuffer` is the byte packet buffer to transfer. Note that odd packets will add the last byte in as an UPPER byte with the lower byte padded with zeros. (ie: last byte is shifted into the upper byte location, rather than left as a lower byte).

```
UINT16 CalCheckSum()
{
        UINT16 n16Checksum=0; UINT16 n16Check;
   for ( UINT32 i=0; i < m_n32PacketSize ; i+=2 )
   {
      n16Check = (UINT8)m_pBuffer[i];
      n16Check = n16Check<<8;
      if ( i+1 != m_n32PacketSize ) n16Check += (UINT8)m_pBuffer[i+1];
      n16Checksum += n16Check;
   }
   return n16Checksum;
}
```

Acknowledge Packet

The Acknowledge (ACK) packet is a general-purpose packet used to *acknowledge* receipt of a packet. The ACK packet does not contain any data other than the packet id it is an ack for. This is sent if the checksum passes, if the checksum fails there is an error message but not an ACK.

ACK Packet		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D10`	10
PacketId	varies	2 (1 UINT16)
CheckSum	0x0B4A	2 (1 UINT16)

4 RIAAT Data

The RIAAT data format is not needed for Command and Control and therefor is not covered in this document. The RIAAT data format is explained in Packet Documentation.

5 Status Packet

Status Request Packet

A Status packet is sent to the host in response to a Status Request Packet. The Status Request packet is defined as follows. The version in the header packet determines which version of the status is returned, 0 or 1.

Status Request Packet		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D01	10
CheckSum	0xFC4A	2 (1 UINT16)

Note, the Status Request packet does not contain any packet data. In the header, set the nVersion to zero to return status packet v0 (PACKET_CC_STATUS_TLM = 0x0D02) or set nVersion to one to return status packet v1 (PACKET_CC_STATUS_TLM2 = 0x0D12).

Status Packet

Status packets serve several purposes: to communicate the status of the SL sensor, to communicate receipt of configuration items, and finally to act as a *heartbeat*. In this fashion, the host must request a status packet, followed by the SL response with a Status packet. As used as a heartbeat, upon socket connection, the host must send status requests at least once every **10 seconds.** Similarly, the SL, once it receives a status request, must respond within **5 seconds.** Typically, a status messages should be sent at approximately 1 Hz for effective command and control.

The Status packet is defined as follows. Please reference the Appendix for details regarding contents of the Status structure.

Status Packet v0		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D02	10
Status Structure		size of status structure
CheckSum	varies	2 (1 UINT16)

Status Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D12	10
Status Structure		size of status structure
CheckSum	varies	2 (1 UINT16)

A feature that has not been implemented yet is the reporting of the last error number or “lastErrorNo”. This was a way to sum up any error messages that might have occurred into the status using an integer to represent the error text information and to reduce the amount of data that needed to be sent from sensor to user. In practice the number of messages is fairly low and this optimization has not yet been done. If non-zero, an error or warning occurred on the SL sensor. The error number will only be reported for one cycle – in other words it will be cleared upon successful Status Packet transfer.

Errors and warnings will be sent FIFO style first with information level messages only being sent if there are no Errors (highest priority) or warnings (medium priority). Information level messages are not queued up and only the last one that occurs will be identified. At a later date, a Table will be produced of errors, warnings and information that identify the number to message mapping.

The pan tilt mode status is a number indicating the current state machine value.

Status Packet ptMode v1		
Item	**Value**	**Description**
STATE_WAIT_FOR_INIT	0	Starting point
STATE_EMPTY_PORT	1	Clearing UART buffers
STATE_INIT_CMD1	2	Command set 1
STATE_INIT_CMD2	3	Command set 2
STATE_INIT_CMD3	4	Command set 3
STATE_INIT_CMD4	5	Command set 4
STATE_DO_STARTUP_CMDS	6	Startup cmds to device
STATE_GET_POSITION_PAN	7	Get pan position
STATE_GET_POSITION_TILT	8	Get tilt position
STATE_HOME_PAN_TILT	9	Homing
STATE_READ_SERIAL_NUMBER	10	Read serial number
STATE_READ_FIRMWARE_NUMBER	11	Read firmware number
STATE_READ_TEMPERATURE1	12	Pan temperature
STATE_READ_TEMPERATURE2	13	Tilt temperature
STATE_FINAL_INIT	14	Done with initialization
STATE_RUN	15	Waiting for a command
STATE_RUN_BUSY	16	Running a command
STATE_STOP1	17	Stop for Pan
STATE_STOP2	18	Stop for Tilt
STATE_DO_STARTUP_ERROR	19	Startup error catch
STATE_READ_FAIL	20	Read failed
STATE_ALL_STOP	21	Done with stop, ready for cmd
STATE_ERROR_REINIT	22	Critical error, re initialize

The Pan Tilt ptSwitches indicates 16 status bits split between active configurations and status of states that have already been done. For instance the **Initializing** bit will go to the value of 1 while the system is going through initialization and when it is finished the value will go to 0 and the **Initialized** field will go to 1 if the sequence was successful.
The fields are used to allow or disallow GUI commands and verify state is ready to accept move commands.

Status Packet ptSwitches v1		
Item	**Bit location**	**Description**
bESTOP	0	Emergency Stop
bSerialBusy	1	Serial busy
bBusyInitializing	2	Initializing
bHome	3	Homed
bReadPos	4	Reading position
bPanStable	5	Pan stable
bTiltStable	6	Tilt stable
bRequestStop	7	Request Stop
bCmdStop_good	8	Accepted Stop command
bPowerState	9	Powered on
bInitialized	10	Initialized
bEnableRotor	11	Power enabled
bCfgRecieved	12	Received Configuration
bICmdsRecieved	13	Received initialization cmds
bError	14	Error state
bStall	15	Stall state

6 Text Packet

Text Packet

A Text packet is sent from the SL Sensor to the host providing textual message, containing general information and /or error and warning information. Text packets can be sent at any time. IF the host is not listening on the UDP channel, then the information is lost. Future development, the status packet will also report the last error number (Not implemented in the current version of the code).

Text Packet		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D06`	10
Error Code	varies	2 (1 UINT16)
Error Level	varies	1 (1 UINT8)
ASCII Test [256]	varies	Variable, up to 256 (max size)
CheckSum	varies	2 (1 UINT16)

Error Level		
Item	**Value**	**Description**
Information	1	
Warning	2	
Error	3	

Error Number chart will be expanded later.

Error Number chart		
Error Code	**Error Level**	**Description**
0x0001	3	Example of an Error message

7 ICD Operation Packet

The Start Operation packet is used to start SL scanning operations with a saved configuration number. The configuration packet is also allowed to start a scanning operation. To end a scan, send the Stop Operations packet. The Start / Stop Operation packet is defined below. When the host sends a Start or Stop Operation packet the SL will respond with an ACK if the checksum passes and then after the command is acted on it will also send a STATUS packet (usually within 5 seconds of starting the command). If the command fails for some reason, the status packet will show that.

The version 1 operation packets added Pan Tilt commands {On, Off, Stop, Zero, Home, send Initialization commands} and a shutdown command (exit the software).

Stop Operation Packet v0 / v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x1000	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x085A / 0x085B	2 (1 UINT16)

Start Operation Packet v0		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0001	2 (1 UINT16)
ConfigNum	varies	2 (1 UINT16)
CheckSum	varies	2 (1 UINT16)

Pan Tilt On Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0010	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x144A	2 (1 UINT16)

Pan Tilt Off Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0100	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x044B	2 (1 UINT16)

Pan Tilt Stop Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0110	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x144B	2 (1 UINT16)

Pan Tilt Zero Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0120	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x244B	2 (1 UINT16)

Pan Tilt Home Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0130	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x344B	2 (1 UINT16)

Pan Tilt Initial Commands Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0x0140	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x444B	2 (1 UINT16)

Shutdown Operation Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: 0x0D0D	10
Command	0xFFFF	2 (1 UINT16)
ConfigNum	0	2 (1 UINT16)
CheckSum	0x0449	2 (1 UINT16)

8 Scan CFG Packet

Scan CFG Packet

The Configuration (CFG) packet is used to communicate a specific SL configuration for scanning operation. All information necessary to perform scanning operations is provided in the CFG packet. The host will send a CFG packet and the SL will respond with an ACK packet if the checksum passes within 1 second. If the nCfgNumber is set to 0, then this will also make the sent configuration active and start a scan which will also send a status packet so that the successful start of scan or failure to start scan can be seen. The active CFG information will be reported in the status messages from the SL Sensor. Please refer to the Appendix for details regarding contents of the ScanCFG structure.

Scan CFG Packet v0		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D03`	10
ScanCFG Structure		size of CFG structure
CheckSum	varies	2 (1 UINT16)

Scan CFG Packet v1		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D11`	10
ScanCFG Structure		size of CFG structure
CheckSum	varies	2 (1 UINT16)

The ScanCFG structure contains an element named “ScanPatternType”. This element is used to define the scan pattern type. Continuous scanning will repeat the configured scan pattern until a Stop packet is received. Single scanning will perform the configured scan pattern only once, and the sensor will automatically stop when pattern finishes. Scanning operations will be communicated via the status record elements “scanInOperation” and “nAutoModeStatus”.

ScanPatternType		
Source	**Value**	**Description**
AZEL_SCAN	0	Take one scan, single
HIGH_RES_AZEL_SCAN	1	Take a detailed scan, single
CONTINUOUS AZ	2	Scan horizontally, repeat
CONTINUOUS_AZEL_FAST	3	Take multiple line scan quickly, repeat
CONTINUOUS_EL	4	Scan vertically, repeat
SIM	5	Data source is simulated, various
STOP_AND_STARE	6	Take detailed scan averaging lots of returns

CTD data can be read from various equipment. A table of those equipment are as below and will be expanded. A Value of 0 means there is no CTD source. Note that sensor will not start for laser safety without depth information.

CTD Sources		
Source	**Value**	**Description**
Extern Laptop CTD	1	
MBARI_TCP	2	
Operator	3	Alternate option, Using CTD Packet
MIDAS_SVX2	4	
NAV_PACKET	5	iXBlue NAV AND CTD
MINI_CTD	6	

INS data can be transferred from various equipment and formats. How to set the Navigation CTD is explained in later chapters.

INS Data Type		
Source	**Value**	**Description**
None	0	No NAV data option
Navigation CTD	1	iXBlue NAV AND CTD
Navigation LONG	2	iXBlue NAV LONG

PPS signal with ZDA packets are used for time synchronization.

Time Synchronization Source		
Source	**Value**	**Description**
None	0	No Synchronization option
TCP_GPS_ZDA	1	TCP/IP packet
GPS_NMEA_ZDA	2	RS-232 Packet

CCI Version 1 Additions

Version 1 of the configuration packet adds in support for Pan Tilt, Wait timer, mount configuration and scripting. The sequence is always pan move, then perform any wait, then scan and last it can also specify the next scan to go to (by number) for an automatic scripting mode of operation. Although the scan can specify all three actions of move, wait, scan, it may also specify only to perform a subset. There are enables for each of the actions.

Mount Scan Type v1		
Source	**Value**	**Description**
ST	0	Static
MO	1	Motion
ZZ	2	None

Mount Pan Type v1		
Source	**Value**	**Description**
PT	0	Sidus Pan and Tilt
PA	1	Sidus Pan Only
3D	2	3D PanTilt
ZZ	3	None

Mount Direction v1		
Source	**Value**	**Description**
PO	0	Port
PD	1	Port Down (45 degrees)
DO	2	Down
SD	3	Starboard Down (45 degrees)
ST	4	Starboard
ZZ	5	None

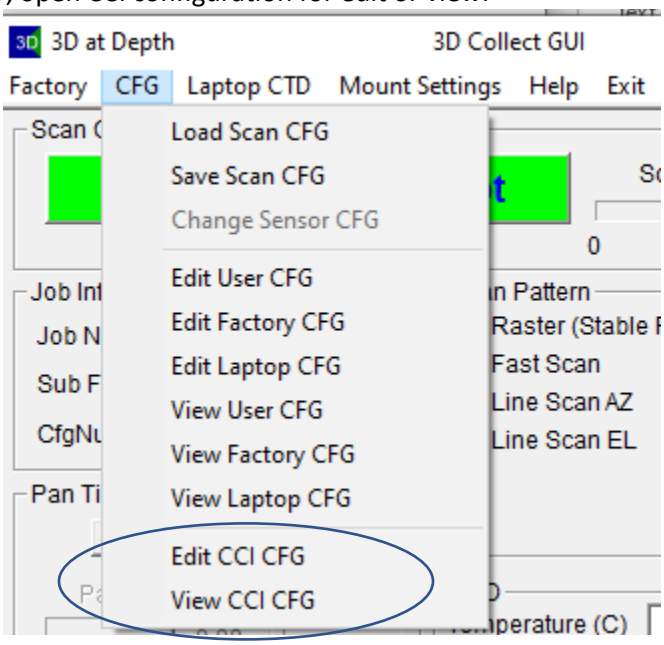
Mount Ins v1		
Source	**Value**	**Description**
PH	0	Phins INS
RO	1	Rovins INS
ZZ	2	None

Mount Collar v1		
Source	**Value**	**Description**
CO	0	Collar
ZZ	1	None

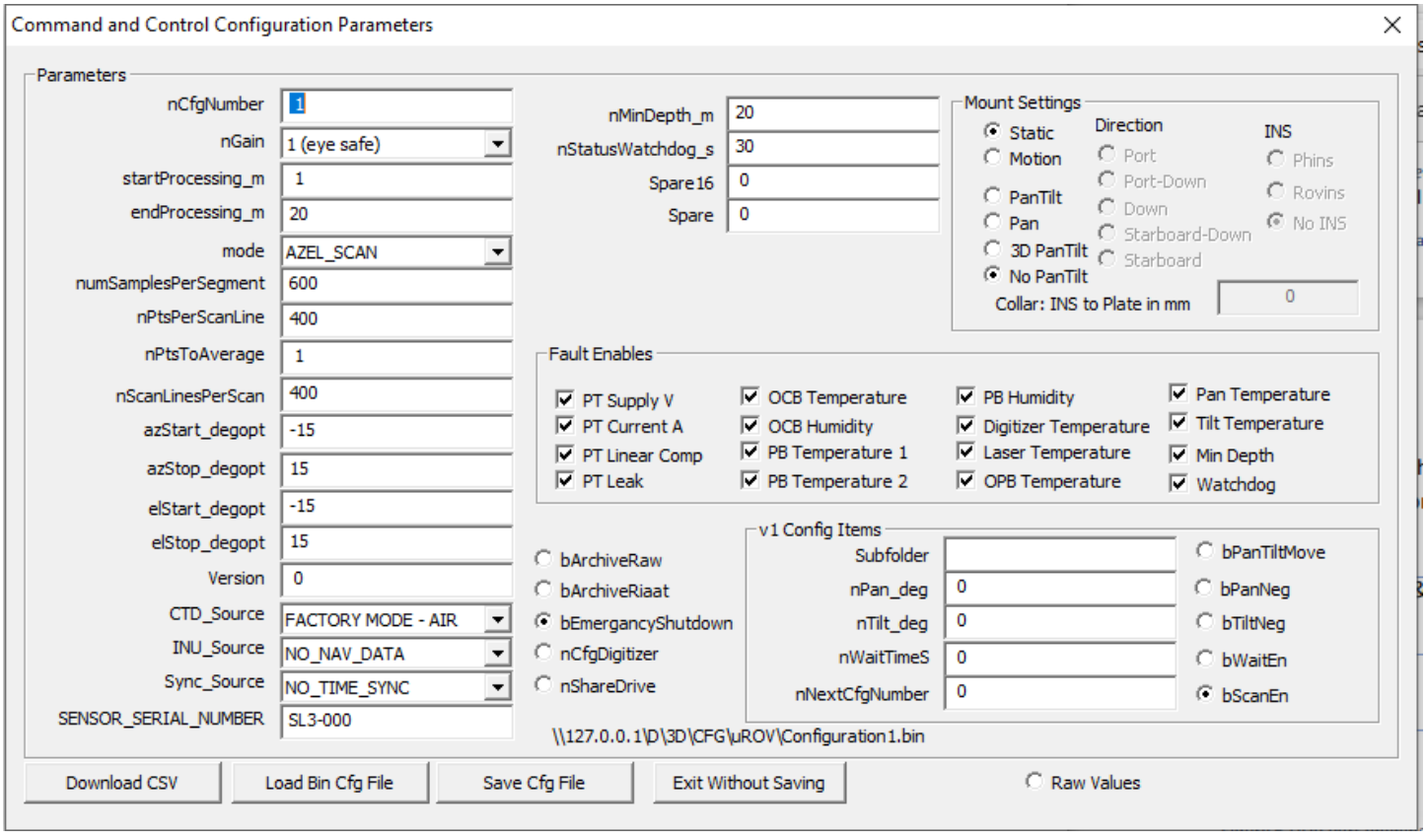
3D at Depth will provide or describe a simple file generator for generating configuration files to send to the SL sensor. The 3D Collect Gui can generate packets and also has a configuration packet viewer/editor for direct inspection or editing of the configurations. The computer running the GUI must have access to the sensor hard drive to bring up the configuration since they are saved on the sensor. Alternatively, the configuration could be copied to the local drive for editing and then copied back.

3D Collect GUI for CCI Configuration Edit or View

Steps for using the viewer/editor, open CCI configuration for edit or view:



Then select edit or view. Below is the edit. Any or all of the fields may be edited. The default is in engineering units but it can also work with raw values. This grant access to all fields of the configuration structure. See the structure definition for appropriate values and options. The drop down menus automatically limit the select to valid values. See the GUI help for additional information on this feature.



9 CTD Packet

CTD Packet

The CTD packet is only needed if the NAV_CTD is not available from the INS. This can also be used during test if an INS system is not attached.

CTD Packet		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D05`	10
CTD Structure		size of structure
CheckSum	varies	2 (1 UINT16)

10 RTV CFG Packet

RTV Configuration packet over UDP.

RTV configuration packet is used to set parameters for the RTV screen. Must be delivered to the RTV SW itself through RTV port on the on-board computer.

RTV CFG Packet		
Item	**Value**	**Bytes**
Packet Header	Packet ID: `0x0D04`	10
Rtv Cfg Structure		size of RTV CFG structure
CheckSum	varies	2 (1 UINT16)

RTV responds with packet **ICDRtvCFGPacket** with the new interpreted RTV configuration (limits applied) but will only do so if the checksum passes.
RTV is based upon x64 architecture.
It will be delivered with either an install package or with a list of required dependencies.

11 Time and INS Setup

The following sections outline some of the expected setup of the Timing / GPS and the INS.

Time Strings and PPS

The ZDA timing string protocol is sent over serial communications. The serial communication, ZDA string, and PPS signal details are shown in the Figure below.

NOTE:
Serial Time String & PPS
Required for moving platform implementations only

PPS Signal Active High
PPSHigh =2.5V to 5V
PPS Low = 0V to 0.8V

Serial Comm Setting: 57.6K Baud, 8N1

NMEA-0183 message: ZDA
UTC day, month, and year
Example of ZDA message string:
$GPZDA,172809.45,12,07,1996,00,00*45

Local time zone not supported, UTC + 00:00
Coordinated Universal Time only

ZDA message fields

Field	Meaning
0	Message ID $GPZDA
1	UTC
2	Day, ranging between 01 and 31
3	Month, ranging between 01 and 12
4	Year
5	Local time zone offset from GMT, ranging from 00 through +/- 13 hours
6	Local time zone offset from GMT, ranging from 00 through 59 minutes
7	The check sum data always begins with *

Figure 2. Serial ZDA protocol and PPS signal specifications.

For information only, the PPS signal pulse from the iXBLue INS is defined below.

4.2.2 OUTPUT PULSES ELECTRICAL CHARACTERISTICS

The following table details the output pulse characteristics.

Table 22 – Output Pulse Electrical Characteristics

	Symbol	Min	Max	Unit
Output voltage at logic low	V_L	0	0.4	V
Output voltage at logic high	V_H	4.8	5	V
Output current	I_{OUT}	-	4	mA

Figure 3. PPS Pulse definition – same as from iXBlue INS.

PHINS NAV AND CTD Output Protocol

The PHINS INS can be set to output a data string called NAV AND CTD. The SL sensor can read this string and will pull CTD data from this string. CTD data is used to describe water salinity, pressure, and temperature. Typically, this data is sent either from the vehicle or the surface, based upon current water conditions. In some implementations, this data is broadcast to multiple sensors on the vehicle. The INS will send a NAV and CTD packet, and no response is required from the SL. The Figure below shows the INS setup screen for the NAV and CTD Output Protocol.

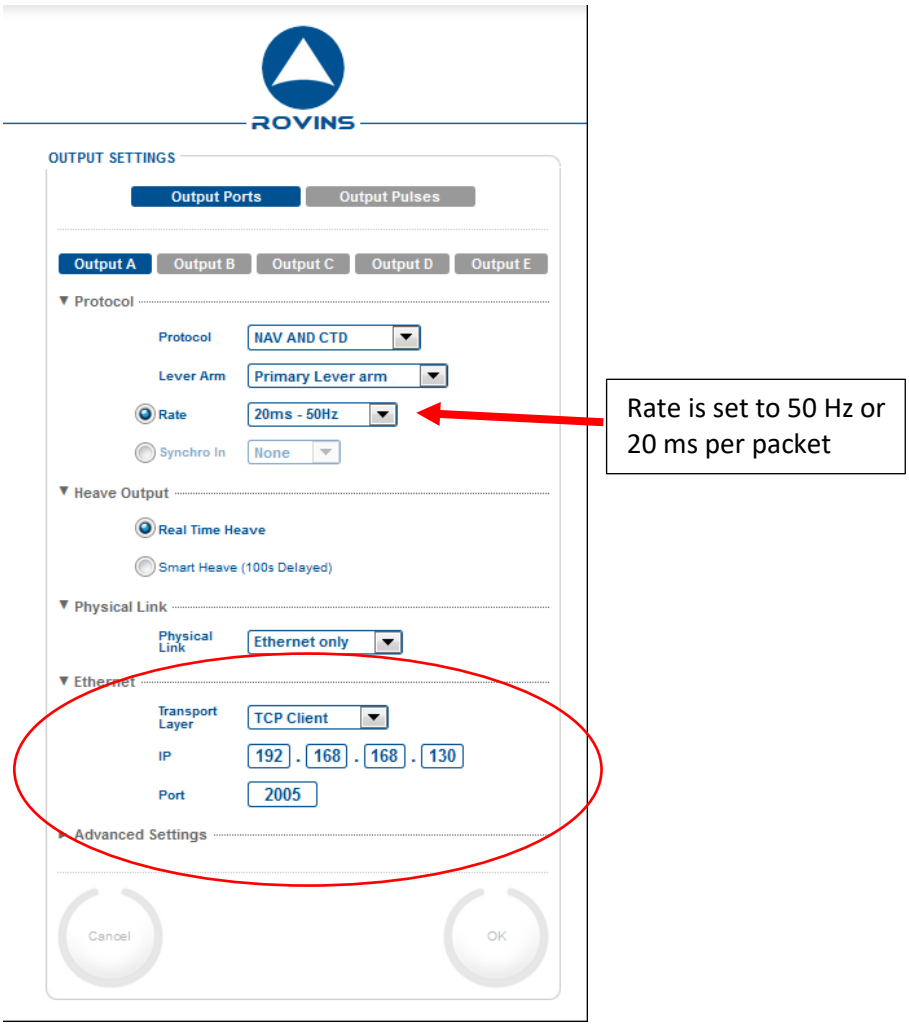


Figure 4. Data Output Port Setup for SubSea Navigation Data Logging – NAV AND CTD. Note this Protocol is on Port 2005 and is the preferred string.

PHINS NAV LONG Output Protocol

The sensor can also accept the NAV LONG output Protocol from the INS. This is provided for information only.

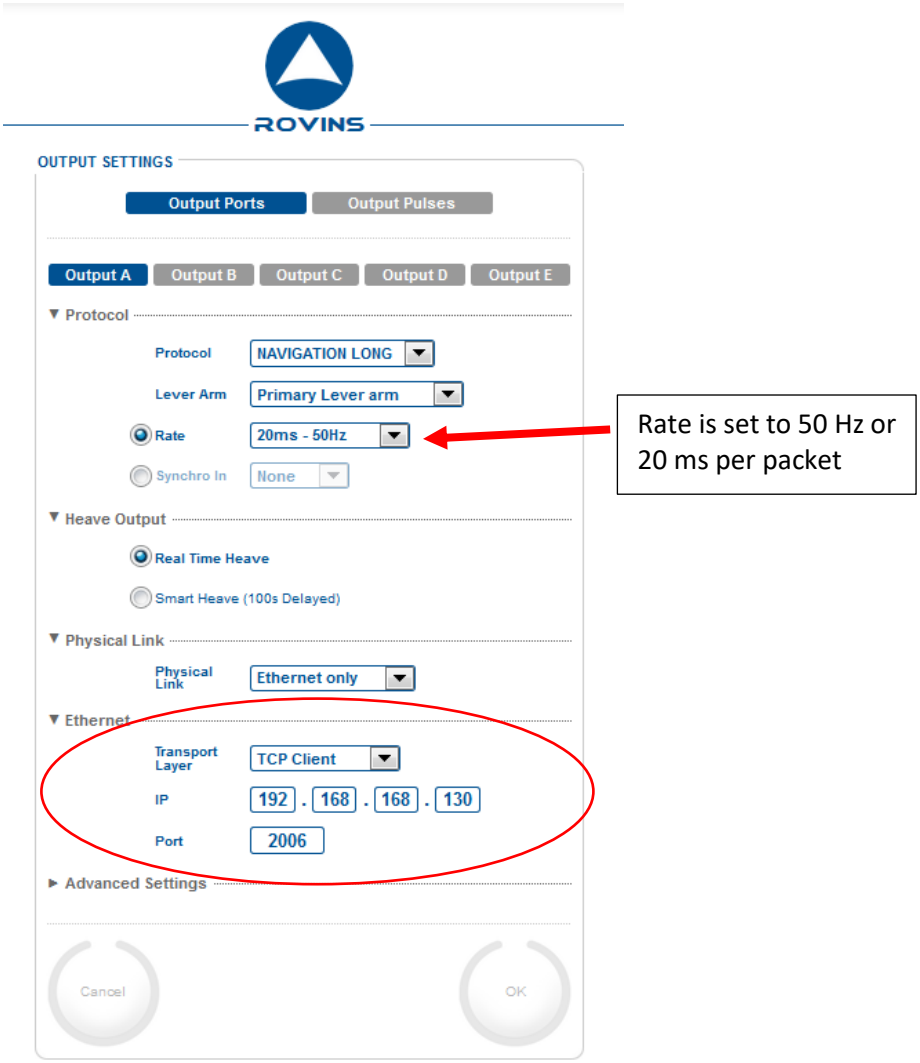


Figure 5. Data Output Port Setup for SubSea Navigation Data Logging – NAVIGATION LONG. Note this Protocol is on Port 2006.

12 General Packet Usage in C++

The data structures and packet definitions were designed for TCP/IP and UDP socket usage. The SL will be configured with a static IP address, and the host will connect to it via standard TCP/IP and UDP socket methods. Several distinct socket connections will exist as follows.

Socket Connections		
SL IP Address	**Description**	**Port Number**
192.168.168.130 (configurable)	CFG, CTD, Start and Stop packets, commands	2010
same as above	Status and ACK telemetry	2011
same as above	Text packet communication	2012
same as above	RIAAT packet communication (TCP/IP) (to RTV)	2002
same as above	GPS_LIKE packet communication (TCP/IP)	2007
same as above	NAV AND CTD packet communication (TCP/IP)	2005
same as above	NAV LONG packet communication (TCP/IP)	2006
AUV Computer	RTV Config (UDP)	2100
AUV Computer	RIAAT packet communication (TCP/IP) (from SL)	2002

In general, socket communications should follow this protocol:

1. A socket is set up.
2. A packet is sent
3. The receiver first reads the header (n bytes)
    a. receiver verifies valid header items
    b. receiver reads remainder of data, per packet type. This may require multiple read cycles.
    c. receiver processes data and sends acknowledgement per protocol (if necessary)
4. If invalid header, error on read, or error on send
    a. receiver may issue warning
    b. receiver may close and reopen socket if multiple back to back errors occur.

In some instances, a “read timeout” error is acceptable per the desired protocol.

To meet protocol time deadlines, it is recommended that each socket connection should be controlled via a separate thread of execution.

13 AUV Laser Safety

Laser safety will be handled at two levels, one at the AUV level and one at the SL sensor level. The below is a first pass that will continue to be refined.

AUV Laser Safety

1) Whenever a CFG packet is received from the surface (which implies scanning is to start as soon as possible), the AUV will then wait to receive the next 10 readings from the depth sensor. If the depth is below 80 meters and are not exactly the same number, then the CFG packet is sent to the SL sensor to start scanning. If the depth is not below 80 meters or the depth number is exactly the same for all readings, the CFG packet is NOT sent to the SL sensor and a warning is sent back to the surface.
2) The AUV receives a heartbeat from the surface communications at least once every TBD minutes. If this heartbeat is lost for longer than XX minutes then the AUV will send the Stop Operation packet to the SL sensor. Scanning cannot resume until communications with the top side is re-established.
3) If two minutes (TBD?) of depth readings are exactly the same, the AUV will assume the depth sensor has faulted and will send the Stop Operation packet to the SL sensor and send an error message to the surface.

SL Laser Safety

1) Upon the receipt of a packet which starts a scan, the SL sensor will check 10 NAV AND CTD packets to read the pressure (depth) and validate they were recent. If the depth is below a configurable defined 80 meters and are not exactly the same number and are less than 30 seconds old, the system will allow scanning to start for laser powers above eye safe. If less than 80m depth or the depth number is exactly the same for all readings, the system will not start unless the Gain is in eye safe range (Gains 1-4), and will send an Error message in the text packet.
2) If a status request is not received by the sensor over a configurable 30 second period, the SL sensor will assume communication has been lost with the AUV and will stop scanning.
3) If two minutes (configurable) of depth readings are exactly the same, the SL will assume the depth sensor has faulted, will stop scanning, and will send an Error message in the text packet.

For Laser activities that are not eyesafe, it is recommended that the control software enforces some password protection when initiating a scan.

14 Appendix

Definitions for standard structures.

```
// ****************************************************************************
// ** Data structures for the SL product line
// ****************************************************************************

typedef struct
{
   UINT16      AlignmentCode;
   UINT16      PacketID;
   UINT16      Version;  // =HDR_VERSION_ICD=0 for original; =HDR_VERSION_ICD_V1=1 for Pan version
   UINT32      DataSize; // Bytes
}
```

ICDStructPacketHeader;

```
typedef struct
{
   UINT16      nCfgNumber;             // unique configuration number identifier
   UINT8       nGain;                  // 1-12
   UINT8       startProcessing_m;      // # = m*255/50
   UINT8       endProcessing_m;        // # = m*255/50
   UINT8       mode;                   // Scan type
   UINT16      numSamplesPerSegment;   // Digitizer samples per shot
   UINT16      nPtsPerScanLine;        // points per scan line
   UINT16      nPtsToAverage;          // Shots to average into on point
   UINT16      nScanLinesPerScan;      // Number of scan lines per scan file
   INT16       azStart_degopt;         // deg = #*30/32767
   INT16       azStop_degopt;          // deg = #*30/32767
   INT16       elStart_degopt;         // deg = #*30/32767
   INT16       elStop_degopt;          // deg = #*30/32767
   UINT8       Version;                // This packet version number = 0
   UINT8       CTD_Source;             // See table for this
   UINT8       INU_Source;             // See table for this
   UINT8       Sync_Source;            // See table for this

   UINT8       bArchiveRaw       :1;   // Archive RAW data (on sensor) - these get large!
   UINT8       bArchiveRiaat     :1;   // Archive RIAAT data (on sensor)
   UINT8       bEmergancyShutdown:1;   // 1 = If error system shuts down. Add explanation
   UINT8       nShareDrive       :1;   // Always 0; 0 = D: 1 = C:  (allow debug)
   UINT8       nCfgDigitizer     :1;   // 0=(default) internal trigger; 1=external
   UINT8       Spare             :3;

   UINT8       SENSOR_SERIAL_NUMBER;    // 0xAB; A = SL#, B = sensor#; Always 0x20
   UINT16      nMinDepth_m;             // Minimum Depth for scan, fault parameter
   UINT16      nStatusWatchdog_s;       // Status request watchdog time in seconds, fault parameter
   UINT16      nFaultEnable;            // Which faults are enabled/disabled see ICDStructFaultCode
   UINT16      Spare16;                 // Room to grow
}
```

ICDStructScanConfig; // v0

```
typedef struct
{
   UINT16      nCfgNumber;             // unique configuration number identifier
   UINT8       nGainEnd : 4;           // 1-12, sets laser power End
   UINT8       nGain    : 4;           // 1-12, sets laser power Start
   UINT8       startProcessing_m;      // # = m*255/50, start processing at this range
   UINT8       endProcessing_m;        // # = m*255/50, end processing at this range
   UINT8       mode;                   // Scan type
   UINT16      numSamplesPerSegment;   // Digitizer samples per shot
   UINT16      nPtsPerScanLine;        // points per scan line
   UINT16      nPtsToAverage;          // Shots to average into one point
   UINT16      nScanLinesPerScan;      // Number of scan lines per scan file (even)
   INT16       azStart_degopt;         // # = deg/30*32767, Az starting angle
   INT16       azStop_degopt;          // # = deg/30*32767, Az ending angle
   INT16       elStart_degopt;         // # = deg/30*32767, El starting angle
   INT16       elStop_degopt;          // # = deg/30*32767, El ending angle
   UINT8       Version;                // This packet version number = 1
   UINT8       CTD_Source;             // See table for this
   UINT8       INU_Source;             // See table for this
   UINT8       Sync_Source;            // See table for this

   UINT8       bArchiveRaw       :1;   // Archive RAW data (on sensor) - these get large!
   UINT8       bArchiveRiaat     :1;   // Archive RIAAT data (on sensor)
   UINT8       bEmergancyShutdown:1;   // 1 = If error system shuts down. 0 = continue on error
   UINT8       nShareDrive       :1;   // Usually 0; 0 = D: 1 = C:  (allow debug)
   UINT8       nCfgDigitizer     :1;   // 0=(default) internal trigger, SL3; 1=external trigger SL2
   UINT8       Spare             :3;

   UINT8       SENSOR_SERIAL_NUMBER;   // 0xAB; A = SL#, B = sensor#; SL2 = 0x20, SL3 = 0x30
   UINT16      nMinDepth_m;            // Minimum Depth for scan, fault parameter
   UINT16      nStatusWatchdog_s;      // Status request watchdog time in seconds, fault parameter
   UINT16      nFaultEnable;           // Which faults are enabled/disabled see ICDStructFaultCode
   UINT16      Spare16;                // Room to grow

   /////////////// v1 ////////////////// Added for PT control and file name //////////////
   UINT8      Subfolder[24];           // subfolder name
                                       // file is: <Time>_<Pan ang>_<Tilt ang>_<Gain>_<row>x<col>.riaat

   UINT32     nPan_deg     : 10;       // # = |deg|/360*511, Pan angle to go to
   UINT32     nTilt_deg    : 10;       // # = |deg|/360*511, Tilt angle to go to
   UINT32     bPanNeg      :  1;       // Pan  direction 1 = negitive, 0 = positive
   UINT32     bTiltNeg     :  1;       // Tilt direction 1 = negitive, 0 = positive
   UINT32     bPanTiltMove :  1;       // 1 = enable move, 0 = skip pan/tilt move, first action
   UINT32     bWaitEn      :  1;       // 1 = enable wait, 0 = skip wait, second action
   UINT32     bScanEn      :  1;       // 1 = enable scan, 0 = skip scan, third action
   UINT32     nWaitTimeS   :  7;       // 0-127 seconds length of time to wait

   UINT32     MountScanType:  2;       //{"ST","MO","ZZ"},  Scan Type Static, Motion, none
   UINT32     MountPanType :  2;       //{"PT","PA","3D", "ZZ"},  Pan Type Sidus Pan&Tilt, Sidus Pan, 3D, none
   UINT32     MountDir     :  3;       //{"PO","PD","DO","SD","ST","ZZ"},  Direction Port, Port-down, down,
                                       // starboard down, starboard, none
   UINT32     MountIns     :  2;       //{"PH","RO","ZZ"},  INS Type Phins, Rovins, none
   UINT32     MountCollar  :  1;       //{"CO","ZZ"}        Collar, none
   UINT32     MountCollar_mm: 10;      // mm of collar
   UINT32     SpareV1      : 12;       // room to grow.

   UINT16     nNextCfgNumber;          // Continue onto next config (script-mode), 0=end.
}
ICDStructScanConfig_v1; // version 1 which added pan tilt, wait, scripting
```

```
typedef struct
{
   UINT8       bColorByIntensity:1; // 1 = Intensity,          0 = Range
   UINT8       bColorScaleGrey  :1; // 1 = Grey,               0 = Color
   UINT8       bLargePixel      :1; // 1 = Make pixels large,  0 = Small pixels
   UINT8       bLogRIAATFile    :1; // 1 = Log RIAAT file,     0 = Do not log RIAAT file
   UINT8       bOpenTCPServer   :1; // 1 = Open TCP Server,    0 = Do not Open TCP Server
   UINT8       Sparebits        :3;

   UINT8       Spare;
   UINT8       nRangeMin;           // Minimum Range in meters (3 min) Cal: #*255/50
   UINT8       nRangeMax;           // Maximum Range in Meters Cal: #*255/50
   float       fIntensityScale;     // Scale to apply to intensity (Truncate in UINT16)
   INT16       fScanWidthMin;       // View distance negative, #*32767/50
   INT16       fScanWidthMax;       // View distance positive, #*32767/50
   UINT8       nLOS;                // Number of points per pulse, 1-5 option
   UINT8       nIntensityFilterMin; // filter points out if below minimum intensity
}
```

ICDStructRTVConfig;

```
// Status data Small
typedef struct
{
   UINT8    version;
   UINT16   nFaultCode;             // Each bit is the status of a fault see ICDStructFaultCode
   UINT16   lastErrorNo;            // Error number that occurred

   UINT16   scanInOperation   :1;   // 0 = Standby,    1 = Scanning
   UINT16   nAutoModeStatus   :1;   // 0 = OFF,        1 = ON
   UINT16   nLaserStatus      :1;   // 0 = OFF,        1 = ON
   UINT16   nPitchRollStatus  :1;   // 0 = OFF,        1 = ON
   UINT16   bSystemInitialized:1;   // 0 = OFF,        1 = ON
   UINT16   nPPSGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nZdaGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nOCBGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nCTDGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nICloudGood       :1;   // 0 = Off/Failed, 1 = good
   UINT16   nRTVGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nNavGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   gain              :4;   // 1-12

   INT16    azScannerPos_deg;       // deg = #*30/32767
   INT16    elScannerPos_deg;       // deg = #*30/32767
   UINT8    nScansCompletePcnt;     // % = # ( 0 to 100)
   INT8     PBTemperature_C;        // C = #/2 (+64 to -64)
   INT8     DiodeTemp_C;            // C = #/2 (+64 to -64)
   INT8     CryTemp_C;              // C = #/2 (+64 to -64)
   INT8     HeatSinkTemp_C;         // C = #/2 (+64 to -64) Electronics
   INT8     LsrHeatSinkTemp_C;      // C = #/2 (+64 to -64) Laser
   INT8     DigTemp_C;              // C = #/2 (+64 to -64)
   INT8     DigCOSTemp_C;           // C = #/2 (+64 to -64)
   INT8     OCBTemp_C;              // C = #/2 (+64 to -64)
   INT8     OPBTemp_C;              // C = #/2 (+64 to -64)
   UINT8    Voltage5;               // V = #*6/256
   UINT8    Voltage3;               // V = #*4/256
   UINT8    Humidity;               // % = #*100/256
   UINT8    ScanPatternType;        // 0-5
   INT16    pitch_deg;              // deg = #*90/32767
   INT16    roll_deg;               // deg = #*90/32767
   INT8     pitchRoll_temperature;  // C = #/2 (+64 to -64)
   INT8     CTD_temperature_c;      // C = #/2 (+64 to -64)
   UINT16   CTD_salinity_psu;       // PSU = #*42/65535
   UINT16   CTD_pressure_dbar;      // dBar = #/13 (5041.15 to 0)
   INT8     PBTemperature2_C;       // C = #/2 (+64 to -64)
   UINT8    PBHumidity;             // % = #*100/256

   // current CFG settings
   ICDStructScanConfig  m_ScanCFG;
}
```

ICDStructStatus;

```
typedef struct
{
   UINT8    version;
   UINT16   nFaultCode;             // Each bit is the status of a fault see ICDStructFaultCode
   UINT16   lastErrorNo;            // Error number that occurred (not implemented)

   UINT16   scanInOperation   :1;   // 0 = Standby,    1 = Scanning
   UINT16   nAutoModeStatus   :1;   // 0 = OFF,        1 = ON
   UINT16   nLaserStatus      :1;   // 0 = OFF,        1 = ON
   UINT16   nPitchRollStatus  :1;   // 0 = OFF,        1 = ON
   UINT16   bSystemInitialized:1;   // 0 = OFF,        1 = ON
   UINT16   nPPSGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nZdaGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nOCBGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nCTDGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nICloudGood       :1;   // 0 = Off/Failed, 1 = good
   UINT16   nRTVGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   nNavGood          :1;   // 0 = Off/Failed, 1 = good
   UINT16   gain              :4;   // 1-12 Laser Gain Current

   INT16    azScannerPos_deg;       // deg = #*30/32767
   INT16    elScannerPos_deg;       // deg = #*30/32767
   UINT8    nScansCompletePcnt;     // % = # ( 0 to 100)
   INT8     PBTemperature_C;        // C = #/2 (+64 to -64)
   INT8     DiodeTemp_C;            // C = #/2 (+64 to -64)
   INT8     CryTemp_C;              // C = #/2 (+64 to -64)
   INT8     HeatSinkTemp_C;         // C = #/2 (+64 to -64) Electronics
   INT8     LsrHeatSinkTemp_C;      // C = #/2 (+64 to -64) Laser
   INT8     DigTemp_C;              // C = #/2 (+64 to -64)
   INT8     DigCOSTemp_C;           // C = #/2 (+64 to -64)
   INT8     OCBTemp_C;              // C = #/2 (+64 to -64)
   INT8     OPBTemp_C;              // C = #/2 (+64 to -64)
   UINT8    Voltage5;               // V = #*6/256
   UINT8    Voltage3;               // V = #*4/256
   UINT8    Humidity;               // % = #*100/256
   UINT8    ScanPatternType;        // 0-5
   INT16    pitch_deg;              // deg = #*90/32767
   INT16    roll_deg;               // deg = #*90/32767
   INT8     pitchRoll_temperature;  // C = #/2 (+64 to -64)
   INT8     CTD_temperature_c;      // C = #/2 (+64 to -64)
   UINT16   CTD_salinity_psu;       // PSU = #*42/65535
   UINT16   CTD_pressure_dbar;      // dBar = #/13 (5041.15 to 0)
   INT8     PBTemperature2_C;       // C = #/2 (+64 to -64)
   UINT8    PBHumidity;             // % = #*100/256

   ////////////////////////////////////////////////////////
   // Unique status items for version 1 (order is after v0)
   UINT8    CommMode:     3;        // mode of the comm thread (pan, wait, scan)
   UINT8    EyeSafe:      1;        // Last eyesafe check passed
   UINT8    PanTiltModel: 2;        // None, Sidus pantilt, Sidus pan, 3D PanTilt
   UINT8    Spare:        2;

   UINT8    nPanTiltStatus;         // MSB Error        = bit 8
                                    //     Move done    = bit 7
                                    //     Moving       = bit 6
                                    //     Homed        = bit 5
                                    //     Homing       = bit 4
                                    //     Initialized  = bit 3
                                    //     Initializing = bit 2
                                    // LSB Power On     = bit 1
   UINT8    ptMode;                 // See Table, Pan state machine value
   UINT16   ptSwitches;             // MSB Stall state                  = bit 15
                                    //     Error state                  = bit 14
                                    //     Received initialization cmds = bit 13
                                    //     Received Configuration       = bit 12
                                    //     Power enabled                = bit 11
                                    //     Initialized                  = bit 10
                                    //     Powered on                   = bit 9
                                    //     Accepted Stop command        = bit 8
                                    //     Request Stop                 = bit 7
                                    //     Tilt stable                  = bit 6
                                    //     Pan stable                   = bit 5
                                    //     Reading position             = bit 4
                                    //     Homed                        = bit 3
                                    //     Initializing                 = bit 2
                                    //     Serial busy                  = bit 1
                                    // LSB Emergency Stop               = bit 0
```

```
   INT16    PanPos_deg;             // deg = #*360/32767
   UINT32   PanBrake    : 1;        // 1 = on, 0 = off
   UINT32   PanActive   : 1;        // 1 = active, 0=off
   UINT32   PanCurrent_A: 6;        // I  = #/63   (1 to 0)
   UINT32   PanSupply_V : 6;        // V  = #*4/63 (4 to 0)
   UINT32   PanComp_mm  : 6;        // mm = #      (63 to 0)
   UINT32   PanLeak_V   : 6;        // V  = #*4/63 (4 to 0)
   UINT32   PanTemp_C   : 6;        // C = # (63 to 0)

   INT16    TiltPos_deg;            // deg = #*360/32767
   UINT32   TiltBrake    : 1;       // 1 = on, 0 = off
   UINT32   TiltActive   : 1;       // 1 = active, 0=off
   UINT32   TiltCurrent_A: 6;       // I  = #/63   (1 to 0)
   UINT32   TiltSupply_V : 6;       // V  = #*4/63 (4 to 0)
   UINT32   TiltComp_mm  : 6;       // mm = #      (63 to 0)
   UINT32   TiltLeak_V   : 6;       // V  = #*4/63 (4 to 0)
   UINT32   TiltTemp_C   : 6;       // C = # (63 to 0)

   UINT16   nFaultEn;               // Each bit is the status of a fault see ICDStructFaultCode
   // current CFG settings
   ICDStructScanConfig_v1  m_ScanCFG;
}
```

ICDStructStatus1;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   ICDStructScanConfig     nData;         // Config structure to save or to start running.
   UINT16                  nCheckSum;
}
```

ICDScanCFGPacket;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   ICDStructScanConfig_v1  nData;         // Config structure to save or to start running.
   UINT16                  nCheckSum;
}
```

ICDScanCFGPacket_v1;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   ICDStructRTVConfig      nData;      // Config structure to save.
   UINT16                  nCheckSum;
}
```

ICDRtvCFGPacket;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   UINT16                  nCommand;   // command set 0x0001 for start, 0x1000 for stop
   UINT16                  nCfgNumber; // nConfigNumber 0 for last config received.
   UINT16                  nCheckSum;
}
```

ICDOperationPacket;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   UINT16                  nErrorCode;
   UINT8                   nLevel;
   char                    strText[256];   // max size is 256, last value is always zero.
   UINT16                  nCheckSum;
}
```

ICDTextPacket;

```
typedef struct             // Only needed if the CTD is not coming from INS.
{
   ICDStructPacketHeader   sHeader;
   float                   temperature_c;
   float                   salinity_psu;
   float                   pressure_dbar;
   UINT16                  nCheckSum;
}
```
ICDCTDPacket;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   ICDStructStatus         sStatus;     // Upon the heartbeat request, small Status returned
   UINT16                  nCheckSum;
}
```
ICDStatusPacket;

```
typedef struct
{
   ICDStructPacketHeader   sHeader;
   ICDStructStatus_v1      sStatus;     // Upon the heartbeat request, small Status returned
   UINT16                  nCheckSum;
}
```
ICDStatusPacket_v1;

```
typedef struct             // Only needed if the CTD is not coming from INS.
{
   ICDStructPacketHeader   sHeader;
   UINT16                  PacketId;
   UINT16                  nCheckSum;
}
```
ICDAcknowledgePacket;

```
typedef union {
   UINT16 word;
   struct {
   UINT16  nFaultOCBTemp_C     :1;
   UINT16  nFaultOCBHumidity   :1;
   UINT16  nFaultPBTemp1_C     :1;
   UINT16  nFaultPBTemp2_C     :1;

   UINT16  nFaultPbHumidity    :1;
   UINT16  nFaultDigTemp_C     :1;
   UINT16  nFaultLaserHsTemp_C :1;
   UINT16  nFaultOpbTemp_C     :1;

   UINT16  nFaultPTSupply_V    :1;
   UINT16  nFaultPTCurrent_A   :1;
   UINT16  nFaultPTLinearComp  :1;
   UINT16  nFaultPTLeak_V      :1;

   UINT16  nFaultPanTemp_C     :1;
   UINT16  nFaultTiltTemp_C    :1;
   UINT16  nFaultWatchdog      :1;
   UINT16  nFaultMinDepth      :1;
   } bits;
}
```
ICDStructFaultCode;