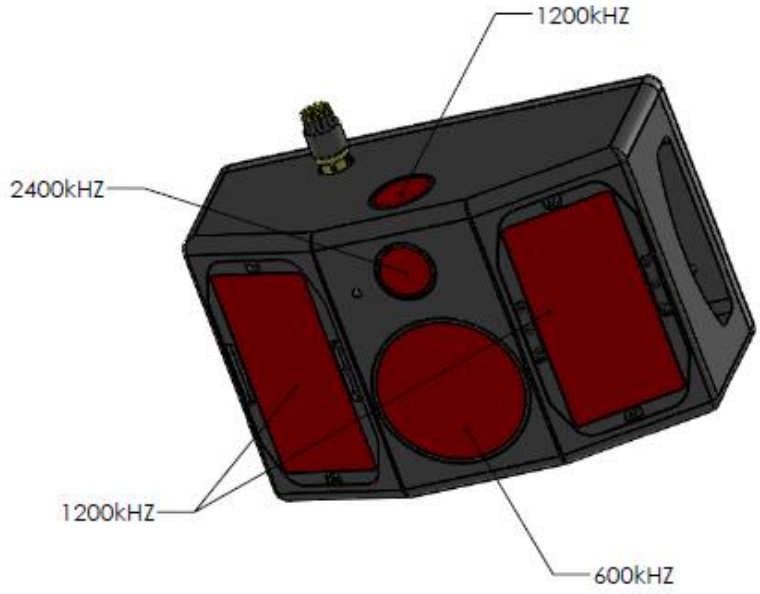
HASCP

HORIZONTAL ACOUSTIC SEDIMENT & CURRENT PROFILER

OPERATION MANUAL



H
A
S
C
P

Rowe Technologies Inc.
12655 Danielson Court,
Suite 306
Poway, CA 92064
USA

Tel: +1 858 842 3020
Fax: +1 858 842 3021

Table of Contents

1 Introduction

Thank you for purchasing a Rowe Technologies Inc. (RTI) HASCP – Horizontal Acoustic Sediment and Current Profiler. This Operation Manual is intended to help HASCP users to get familiar with their system. This manual is specific for using the HASCP. The manual does not discuss all the technical issues of the HASCP. All documentation is being provided to you on USB storage device in a fully searchable, printable, electronic format.

RTI ONLINE

On our website at www.rowetechinc.com, you can also find technical support, user manuals, technical brochures, product datasheet about our other products etc.

1.1 How to Contact Rowe Technologies, Inc.

If you have technical problems with the instrument, please feel free to contact us at:

Rowe Technologies, Inc.

12655 Danielson Court, Suite 306
Poway, CA 92064
USA

Tel : +1 858 842 3020
Fax : +1 858 842 3021
Email : sales@rowetechinc.com
Web : http://rowetechinc.com/

1.2 Inventory check

Check to make sure you received the required content in the received package. Contact RTI if you find any part missing.

1.3 Safety Precautions

2 System Overview

RTI is pleased to introduce the HASCP – Horizontal Acoustic Sediment and Current Profiler. The HASCP operates at a 3 different frequency configuration (600 kHz, 1200 kHz and 2.4 MHz) for sediment and velocity profiling using Narrow band and Broad band operation developed for the needs of scientists and engineers. The HASCP system contains the following,

- Two horizontal 1200 kHz dual beam inclined transducers are rectangular to provide narrow 2-way beam width (0.5 degree) (Beams 1 and 3).
- The 600 kHz and 2.4 MHz (Beam 2 and 4) will be used to be used for precision measurement of the acoustic echo sediment characteristics. The 2-way beam width of the 600 kHz and the 2.4 MHz transducer is 1.1 degrees.
- An additional vertical beam (1200 kHz) is used for measuring the depth of the instrument from the water surface (Beam 5).
- A modern technology multi-beam and multiple frequency ADCP product platform to provide the core HASCP capability within a single package.
- HASCP programming software for user operation setup, data collection.

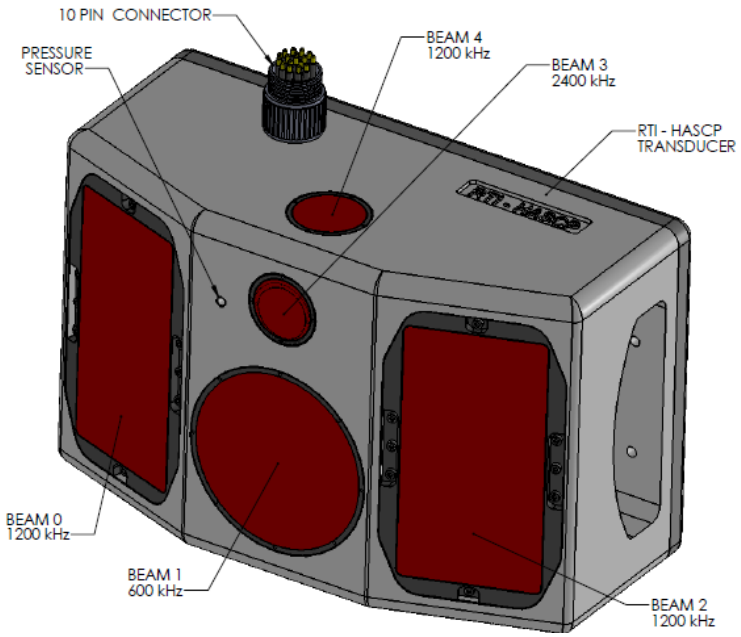


Figure 1. HASCP with the hardware configuration

2.1 Summary of RTI HASCP

Table 1. Summary of RTI HASCP

Operating Frequency	1152 kHz, 576 kHz, 2400 kHz
Operating Depth	100 m
Housing Material	Acetal
Hardware	• Two rectangular transducers @ 1152 kHz for velocity and sediment measurements • A single 4inch Piston transducer @ 576 kHz for sediment measurements • A single 1inch Piston transducer @ 2400 kHz for sediment measurements • A single 1.4 inch @ 1152 kHz Piston transducer for depth measurements • Temperature sensor • Pressure Sensor • 10 pin connector
Communication Configurations	1 RS 485 channels 1 RS 232 channel Trigger out
Cables	5 m Communication and power cable
Internal Storage	32 GB

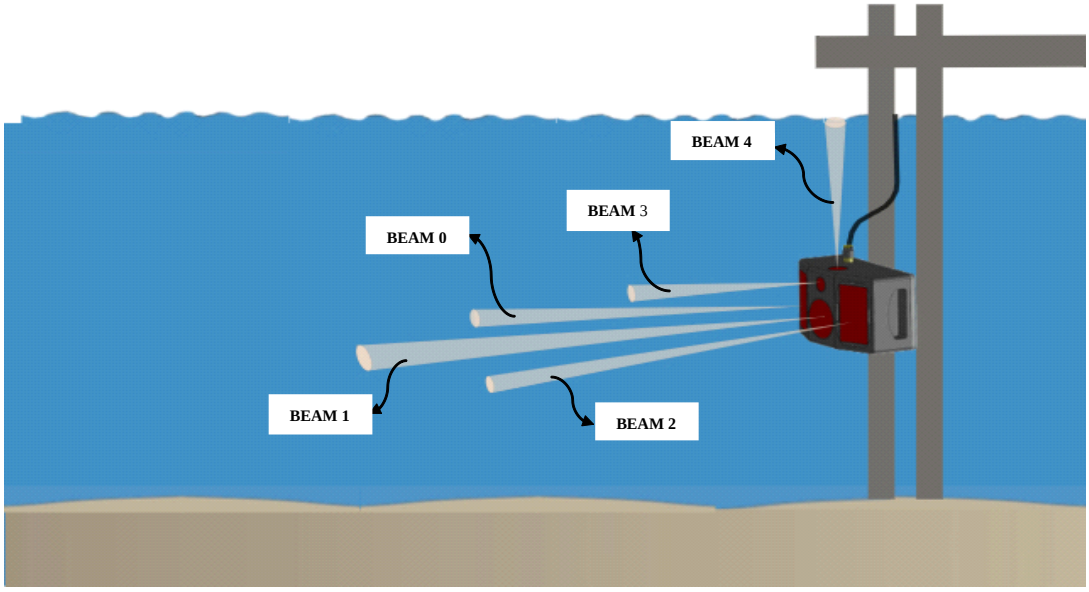


Figure 2. Illustration of the various acoustic beams from the HASCP unit. Beams 0 and Beam 1 are rectangular transducers with narrow beam width (0.5 degrees) and operate at 1152 kHz. Beam 2 and Beam 3 operate at 576 kHz and 2304 kHz respectively. Beam 4 operates at 1152 kHz and is used for measuring the depth of the instrument from the surface.

2.2 Connections to the instrument

Table 2. Connections on the HASCP

Connections

[Communication cable to HASCP unit]
Make sure to properly orient the pins on the male and female connectors.

[Communication cable Serial Interface and Power]

Trigger out, DAT_COM, Shield

RS-485

RS-232

Power

Note: *Apply Molykote 44 to female part of the underwater connector before mating the underwater connector. The link from the vendor is given below for reference.*

Link : https://www.macartney.com/what-we-offer/systems-and-products/connectors/subconn/subconn-general-technical-information/subconn-handling-instructions/

To communicate to the external world through RS 485, or RS 232, plug in the female part of the connector of the 5 m connection cable into the HASCP housing. The other end of the communication cable has the following interfaces,

Table 3. HASCP communication interfaces

RS 232 (Standard) DB9	a. 8 bit, No Parity, 115200 Baud (can go higher for short distances) b. Full Duplex (3 wire, shares return with 485) c. Minimal noise immunity d. Short distances (20 m) e. Single ended - unbalanced
RS-485 (Standard) Terminal block	a. 8 bit, No parity, up to 921600 Baud b. Half Duplex (3 wire, shares return with 232) c. Good noise immunity d. Long distances (1000 m) e. Differential – Balanced – one line true signal and other line is false signal or inverted signal. The receiver detects the input and sees which signal is more positive than other.
Power	9V – 24V
TRIG Out	Trigger output line from HASCP unit
CHGND	Chassis Ground line (used in case of EMI issues)
TRIG IN	Not available and is intended for future use

3 Getting Started

This chapter details the connecting to the instrument, installation of the software and working with the instrument. We strongly recommend you read all of the provided documentation to learn the full capabilities of your HASCP.

3.1 Connecting to an HASCP

The procedure for communicating to an HASCP is described below. The steps are listed below sequentially and RTI recommends that the user follows the steps below in this order.

Note: The RTI HASCP Software is designed for a Windows operating system.

3.1.a Step1: Installing Driver

The first step before connecting to the HASCP is to make sure that the driver for the serial communications between the between the PC and the HASCP is installed. The driver can be found on the small CD provided in the shipping case or at the following link:

http://www.bb-elec.com/getattachment/c8461811-bebf-456a-8386-6ea1281219b4/USB_Drivers_PKG_v2-08-28.zip.aspx

Follow the instructions provided on the screen to install the driver.

3.1.b Step 2: Verify COM port

To verify, insert the USB - Serial Converter into a USB port. Next go to the Control Panel and in the Device Manager menu expand the Ports menu to the COM port. The USB to serial converter should be identified as the following:

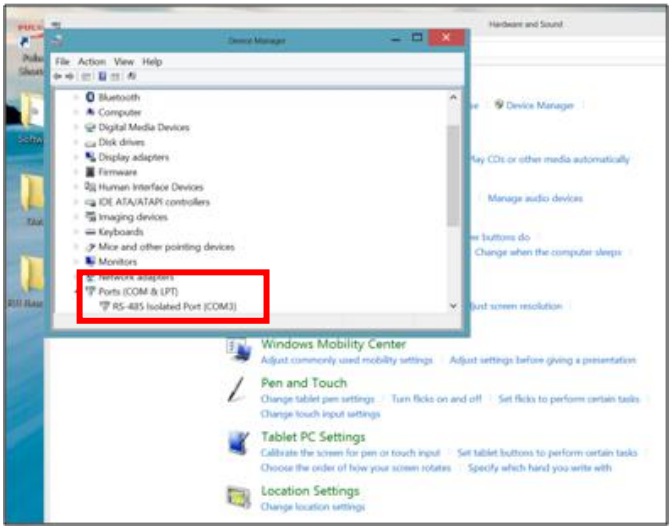


Figure 3. RS 485 Isolated COM Port setup.

The port will be assigned automatically such as COM3. This is indicated by the red box in the screen shot above. This is the COM port that the users should use to connect to the HASCP.

3.1.c **Step 3**: HASCP Interconnecting

Connect the keyed green five-pin connector on the instrument cable pig-tail to the USB-Serial adaptor

Insert the USB connector into a USB port on the PC

3.1.d **Step 4**. Connecting to the Instrument via Software

Once all of the instrument interconnections have taken place, (presented in Step 3) open the RTI HASCP Software to communicate with the ADCP.

3.2 Communications

i. Click the Communications TAB on the top left of the software window. Refer to Figure 4.
ii. If the HASCP is connected and outputting data, the text page on the right will show what data in being transferred. If not, the text page will be blank.
iii. To change the port settings:
    1. Select the communications port which is connected to the HASCP.
    2. Select Baud Rate, number of Bits, Parity, and Stop Bits.

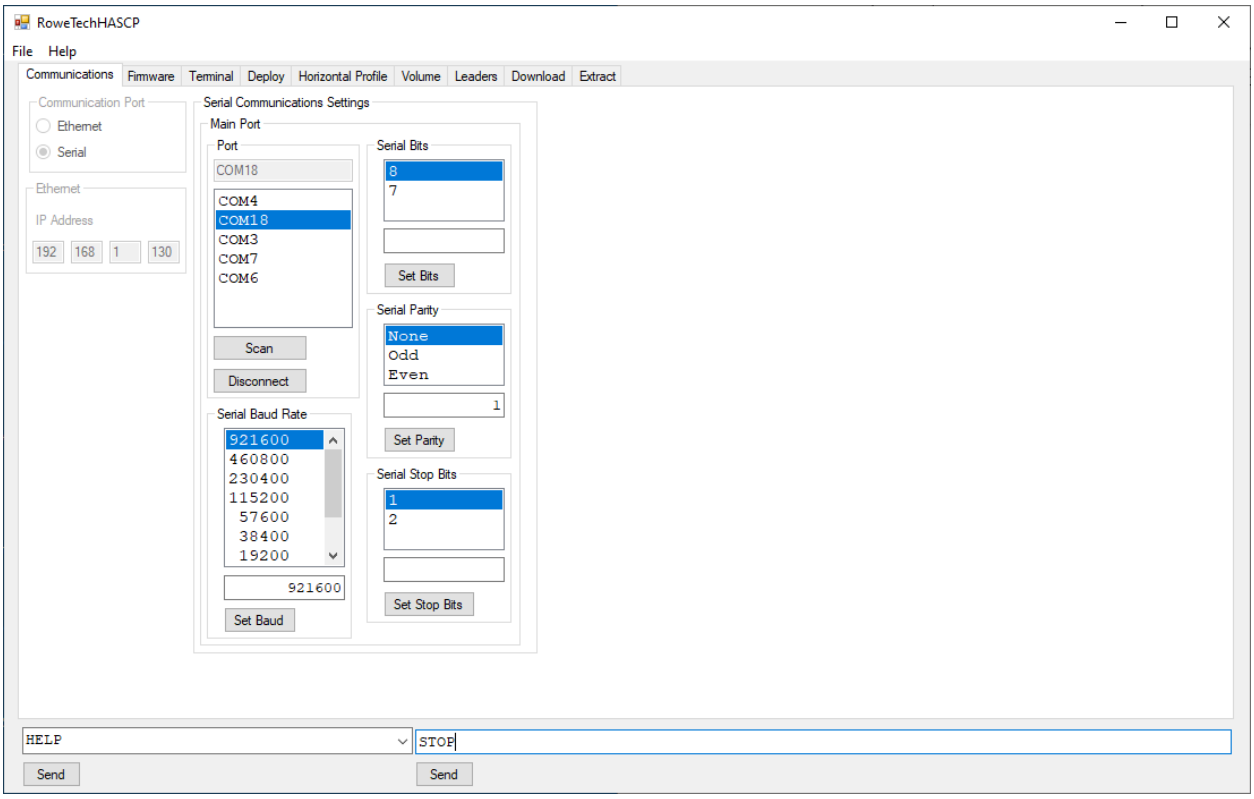


Figure 4. Connecting to HASCP – Communications set up.

3.3 Terminal

a. Click the Terminal TAB near the top left of the software window. Refer to Figure 5.
b. If the HASCP is connected and outputting data, the text page on the right will show what looks like random binary data. If not, the text page contains the last data displayed before the HASCP was stopped.
    i. HASCP Control
        a. To STOP the HASCP click BREAK then, after the Wakeup message appears, click STOP

Copyright (c) 2019 Rowe Technologies Inc. All rights reserved.
HASCP
DP1200 DP600 DP2400 DP1200
SN: 08BJHQ00000000000000000000888888
FW: 00.07.27 Nov 20 2019 05:22:14
STOP+

Troubleshooting: In the case the system does not connect using the known COM port, please try another baud rate. Although, default set-up baud rate is 115200, the system may have been used in another baud rate – You may have to cycle through various baud rates in the COM port to connect to the instrument.

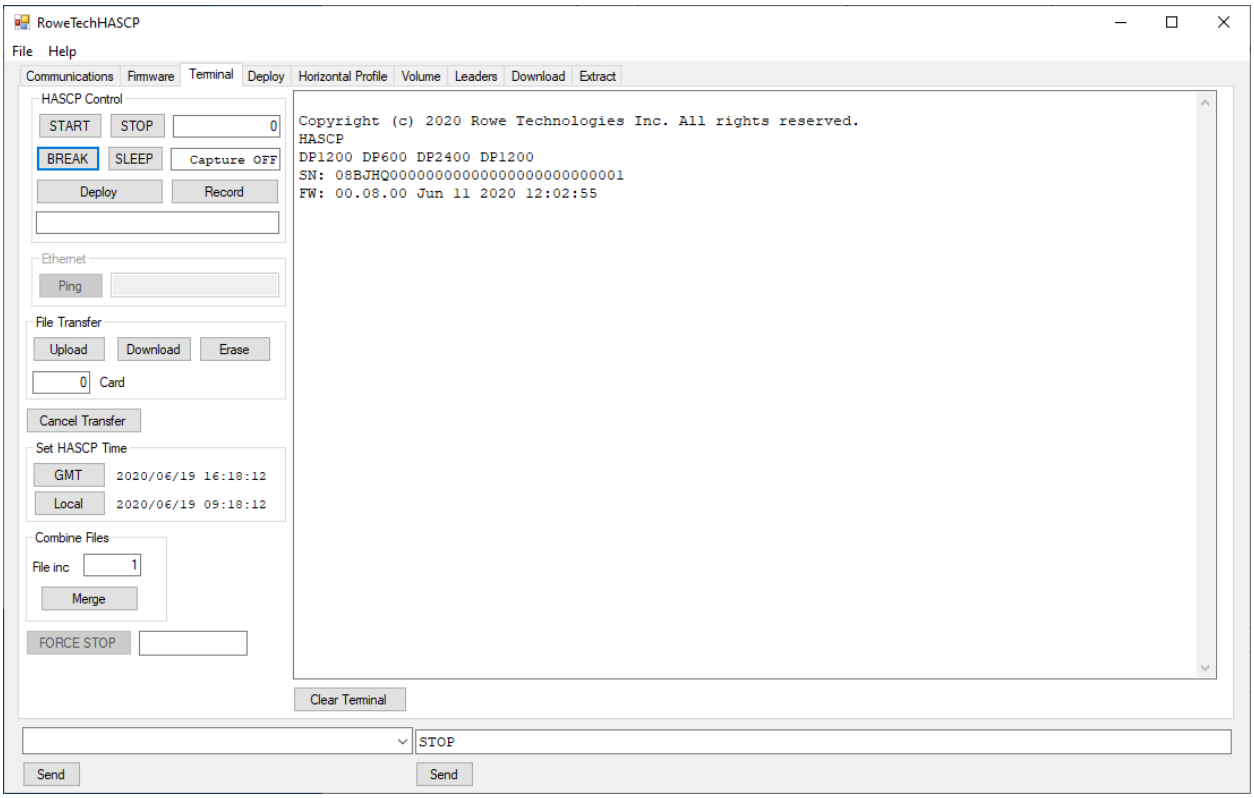


Figure 5. Connecting to HASCP – Terminal TAB showing BREAK message.

b. To START the HASCP click START. The HASCP will begin pinging and outputting ensemble data beginning with ensemble number 1.
c. If you want to send a predetermined set of commands to the HASCP click Deploy and follow the instructions to select the command text file.

c. Set HASCP Time
   i. Click either GMT or Local. Be sure the HASCP is STOPPED before setting the time or the next ping time might be delayed for many hours.
d. File Transfer
   i. Click Upload to transfer a file from the PC to the HASCP SD card.
   ii. Click Download to transfer a file from the HASCP to the PC.
      1. The Text Box at the bottom right of the window must contain the name of the desired file to transfer prior to clicking Download.
   iii. Click Erase to delete a file from the HASCP SD card.
      2. The Text Box at the bottom right of the window must contain the name of the desired file to erase prior to clicking Erase.
   iv. If you wish to cancel the transfer click Cancel Transfer.
e. Record
   i. Clicking Record will turn on/off the HASCP output data recording on the PC.
      1. The text box to the right of the Record button show the number of bytes recorded along with a flashing ON/on. Clicking Record a second time will turn of data recording.
      2. Recorded data is stored at "C:\RoweTechHASCP\Capture". File size is limited to 10 Mbytes. When the 10MB limit is reached the file name will increment by one and recording will continue
f. Combine Files
   i. If you desire to merge files into a single larger file click Merge and follow the instructions.
g. Clear Terminal
   i. To clear the terminal screen click Clear Terminal near the bottom of the window.
h. HASCP Commands
   i. To send single commands to the HASCP type the command in the combo text box at the bottom left then click Send. The combo text box remembers the commands if you press Enter instead of clicking Send.

3.4 Deploy

a. Click the Deploy TAB near the top left of the software window. Refer to Figure 7.
b. Deployment Summary
   i. Ensembles
      1. Total number of ensembles that will be output/recorded during the deployment.
   ii. Min Interval (s)
      1. Minimum time between ensembles to maintain ping timing.
   iii. Data Bytes
      2. Number of bytes that will be output/recorded during the deployment
c. Deployment Information
   i. Days
      1. The expected number of days the deployment will last
   ii. Ensemble Interval (s)

1. The desired ensemble interval. If the ping timing exceed the Ensemble interval the ensemble timing will slip.

iii. Latitude (deg)
   1. Deployment location

iv. Longitude (deg)
   1. Deployment Location

v. Right Bank
   1. Sets Right Bank Deployment location

vi. Left Bank
   1. Sets Left Bank Deployment location

vii. Height above Bottom (m)
   1. Physical location of HASCP above the bottom during deployment
   2. Required for waves calculations

viii. Water Depth (m)
   1. Physical location of HASCP during deployment
   2. Useful for speed of sound calculation if pressure sensor and/or surface detection not available.

ix. Water Temperature (C)
   1. Used when built in temperature sensor is not available.

x. Water Speed of Sound (m/s)
   2. Used for speed of sound when no sensors are available to calculate or measure the speed of sound. Sometimes used when the end user wants to scale the Doppler and range data themselves.

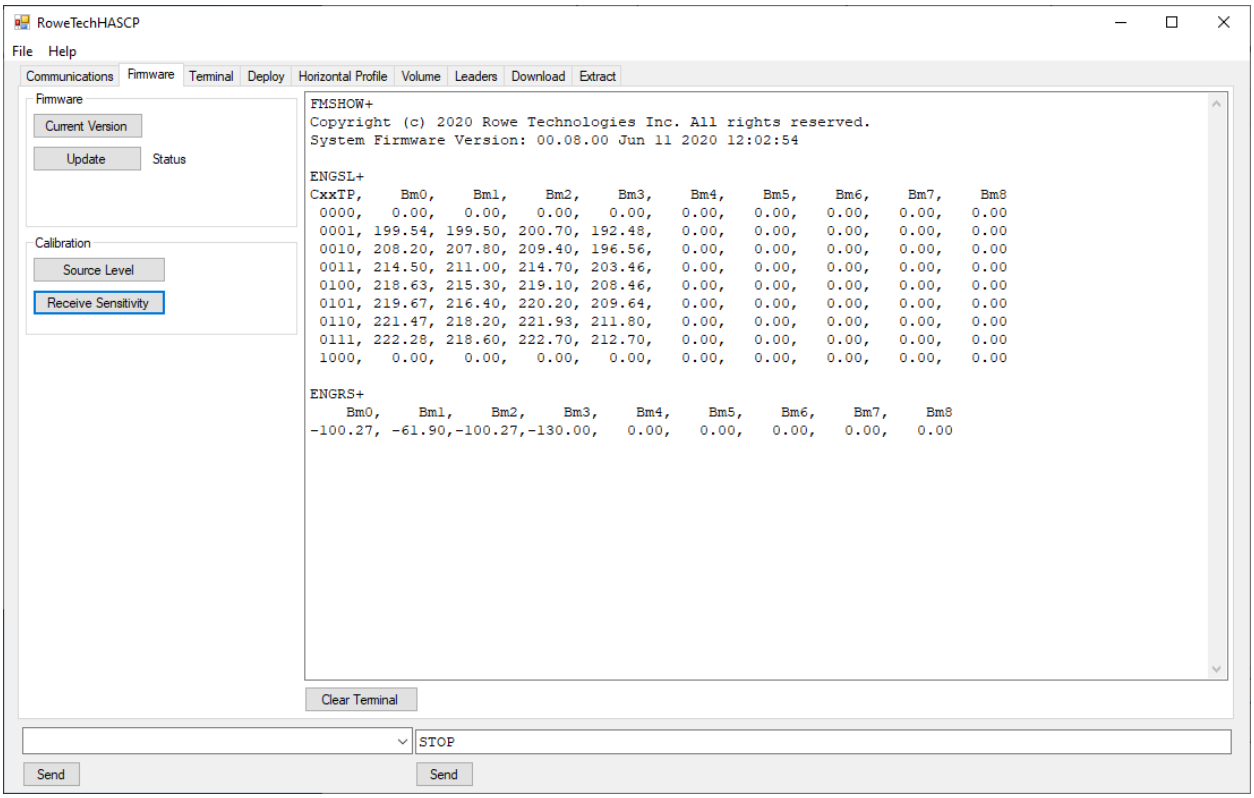


Figure 6. Firmware tab showing the firmware version, source level in dB for different power level, and the receive sensitivity of the HASCP. Bm0 and Bm2 are beam 0 and beam 2 (1200 kHz), Bm1 is beam 1 (600 kHz), Bm3 is beam 3 (2400 kHz) and Bm4 is beam 4 (1200 kHz).

3.5 Deployment Data and Power Options

i. Auto Start on Power Up
   1. Start Pinging and outputting/recording data when power is applied to the HASCP .

ii. Internal Data recording
   1. Turn on/off internal data recording to the SD card.

iii. External Data Logger
   1. Enable the output trigger line to maintain a high level during serial data output. This is useful when the RTI data logger needs to be power cycled.

iv. RS485 Serial Data Output
   1. Enable HASCP data output on the RS485 data lines.

v. RS232 Serial Data Output
   1. Enable HASCP data output on the RS232 data line.

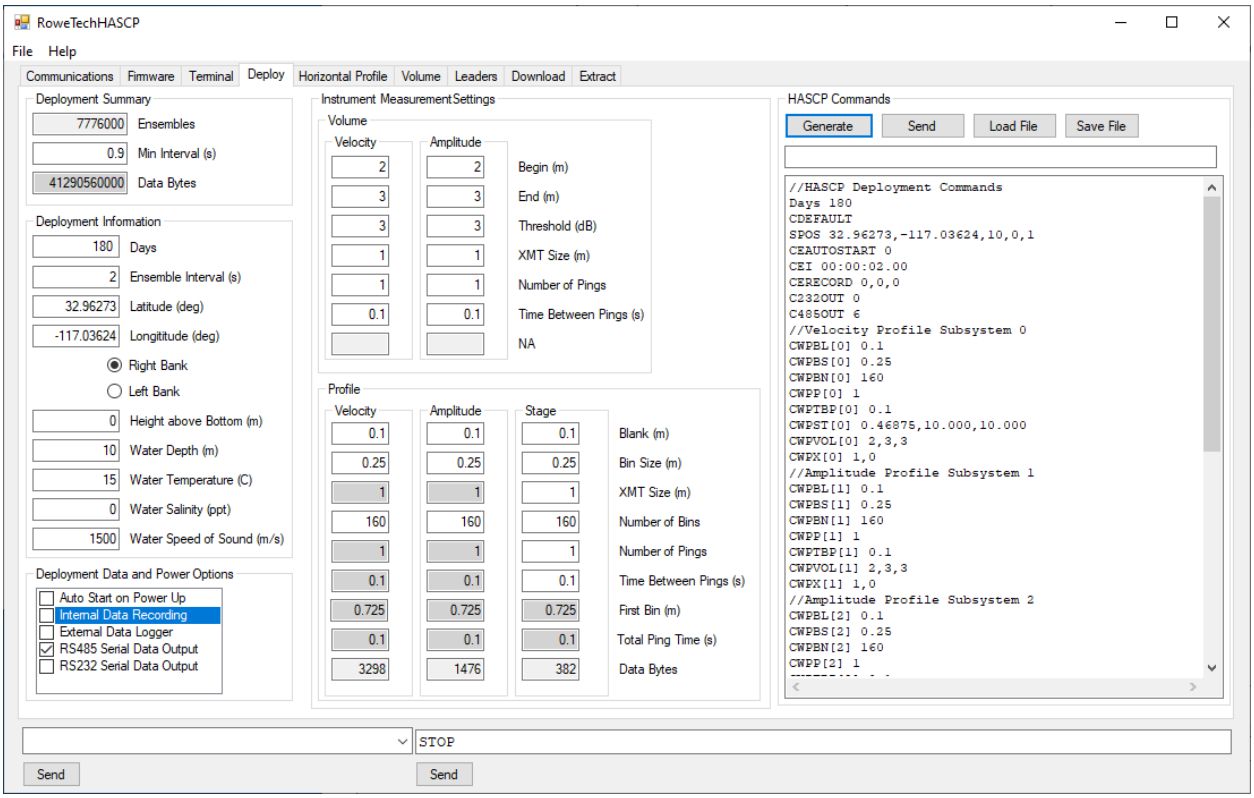


Figure 7. Deploy Page summary showing an example setup

3.6 Instrument Measurement Settings

i. Volume
   1. Velocity and Amplitude
      a. Begin (m)
         i. Selects the first sample location for the volume.
      b. End (m)
         i. Selects the last sample location for the volume.

ii. The End will be automatically moved back towards the Begin if the signal to noise is below the selected threshold.

c. Threshold (dB)
   i. Sets the Signal to Noise threshold for "good" data.
   ii. Setting large negative threshold will turn off the threshold test.

d. Xmt Size (m)
   i. Sets the length of the transmitted acoustic pulse.

e. Number of Pings
   i. Set the number of pings to average together during an ensemble.

f. Time Between Pings (s)
   i. Sets the time between pings which allows for reverberation die down.

ii. Profile

1. Velocity, Amplitude, and Stage

a. Blank (m)
   i. Sets the location of the first sample in the first bin.

b. Xmt Size (m)
   i. Sets the length of the transmitted acoustic pulse.

c. Number of Bins
   i. Sets the number bins in the profile

d. Number of Pings
   i. Set the number of pings to average together during an ensemble.

e. Time Between Pings (s)
   i. Sets the time between pings which allows for reverberation die down.

f. First Bin (m)
   i. Displays the location of the middle of the first bin.

g. Total Ping Time (s)
   i. Displays to the total time the selected pings will take during each ensemble.

h. Data Bytes
   i. Displays the number of data bytes generated during the ensemble.

iii. HASCP commands

1. Generate
   a. Clicking the Generate Button will convert the Instrument Measurement Settings to HASCP commands. The command list is displayed in the Command List Text Box.
2. Send
   a. Causes the Command list in the Command List Text Box to be sent to the HASCP via a serial port.
3. Load File
   a. Allows selection of a previously saved command list.
4. Save file
   a. Saves a copy of the Command List Text Box to a text file on the PC.
5. Information Text Box

a. Shows status and gives error messages that occur during communication with the HASCP.

6. Command List Text Box
   a. Displays the current command set.

3.7 Horizontal Profile

i. Click the Horizontal Profile TAB near the top center-left of the software window. Refer to Figure 8 and Figure 9.

ii. Playback
   1. Select File
      a. Allows selection of previously recorded/saved HADCP data file.
   2. > Playback
      b. Clicking > will cause the software to playback through the selected file
   3. || Pause
      c. Clicking || will pause the Play function. Clicking > will resume playback.
   4. – Step Back one ensemble
      d. When playback is paused the user can click back one ensemble at a time. The ensemble buffer is limited to how many ensembles can be stepped back during a pause.
   5. + Step forward one ensemble
      e. When playback is paused the user click forward one ensemble at a time.
   6. Stop
      f. Stops playback and close the selected playback file.

iii. View
   1. Graph
      a. Selects a graphical view of the profile data.
   2. Text
      b. Selects a text column view of the profile data.
   3. Font/Line Size

iv. Vel Data (Broadband 2 beam Janus)
   1. Beam
      a. Selects the beam velocity data to be displayed.
   2. Inst
      b. Selects the XY velocity data to be displayed.
   3. Amp
      c. Selects the beam amplitude data to be displayed.
   4. Cor
      d. Selects the beam correlation data to be displayed.
   5. Pings
      e. Selects the number of good pings for each beam collected during the ensemble to be displayed. Shown on the text page.

v. Vel Stats (Broadband 2 beam Janus)
   1. Beam
      a. Selects the average beam velocity data to be displayed.
   2. Inst

a. Selects the average XY velocity data to be displayed.

3. Amp

a. Selects the average amplitude data to be displayed.

4. Cor

b. Selects the average correlation data to be displayed.

5. Clear

c. Clears the average data. Allowing restart of the ensemble averaging at the current ensemble.

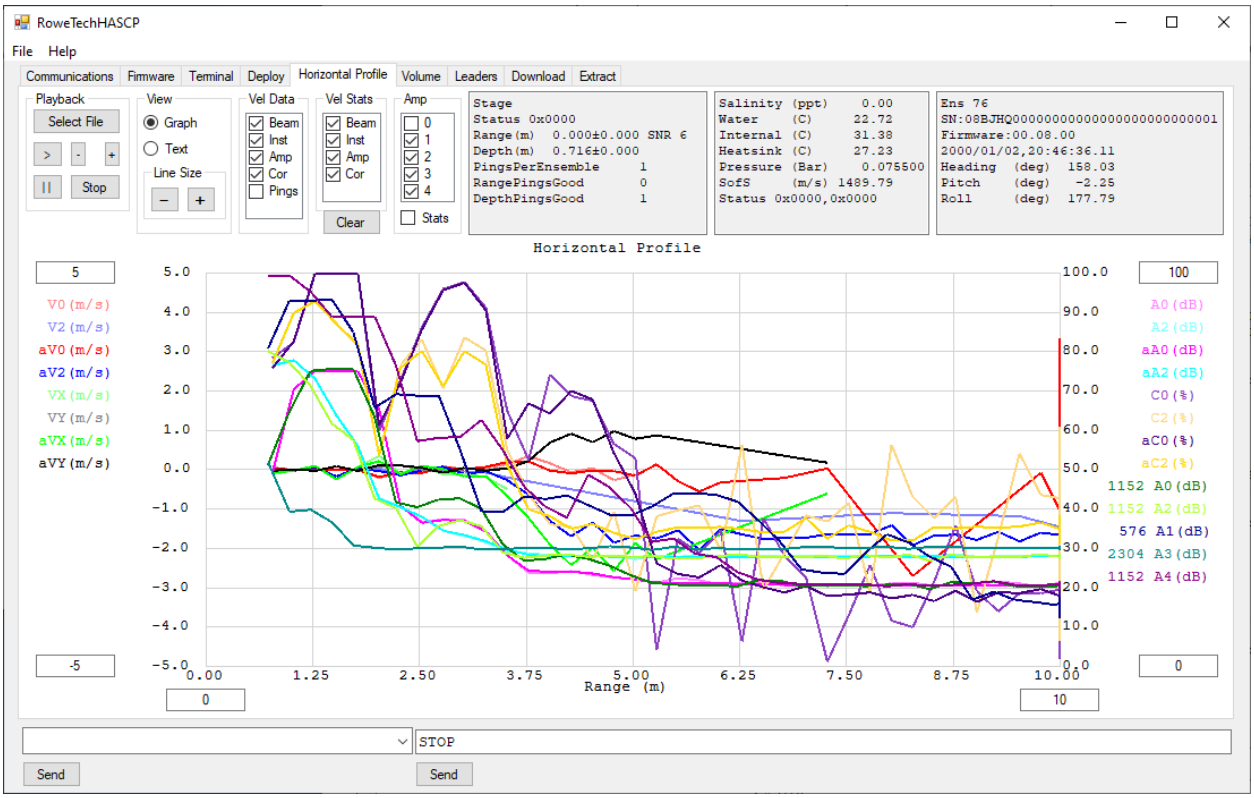


Figure 8. Horizontal Profile display page showing the various profile and amplitudes measured by HASCP. . This data was measured by the HASCP at the lake. The left hand axis is the velocity range (-5 to + 5 m/s) and the right hand axis represents the RSSI measured by the ADCP from 0 to 100 dB. The numbers on the axis can be changed by typing into the box.

vi. Amp (un-coded acoustic transmit-pulse)

1. 0

a. Unused in this configuration

2. 1

b. Selects the 1152 kHz two beam Janus amplitude profiles to be displayed.

3. 2

c. Selects the single beam 576 kHz amplitude profile to be displayed.

4. 3

d. Selects the single beam 2304 kHz amplitude profile to be displayed.

5. 4

e. Selects the single stage beam 1152 kHz amplitude profile to be displayed.

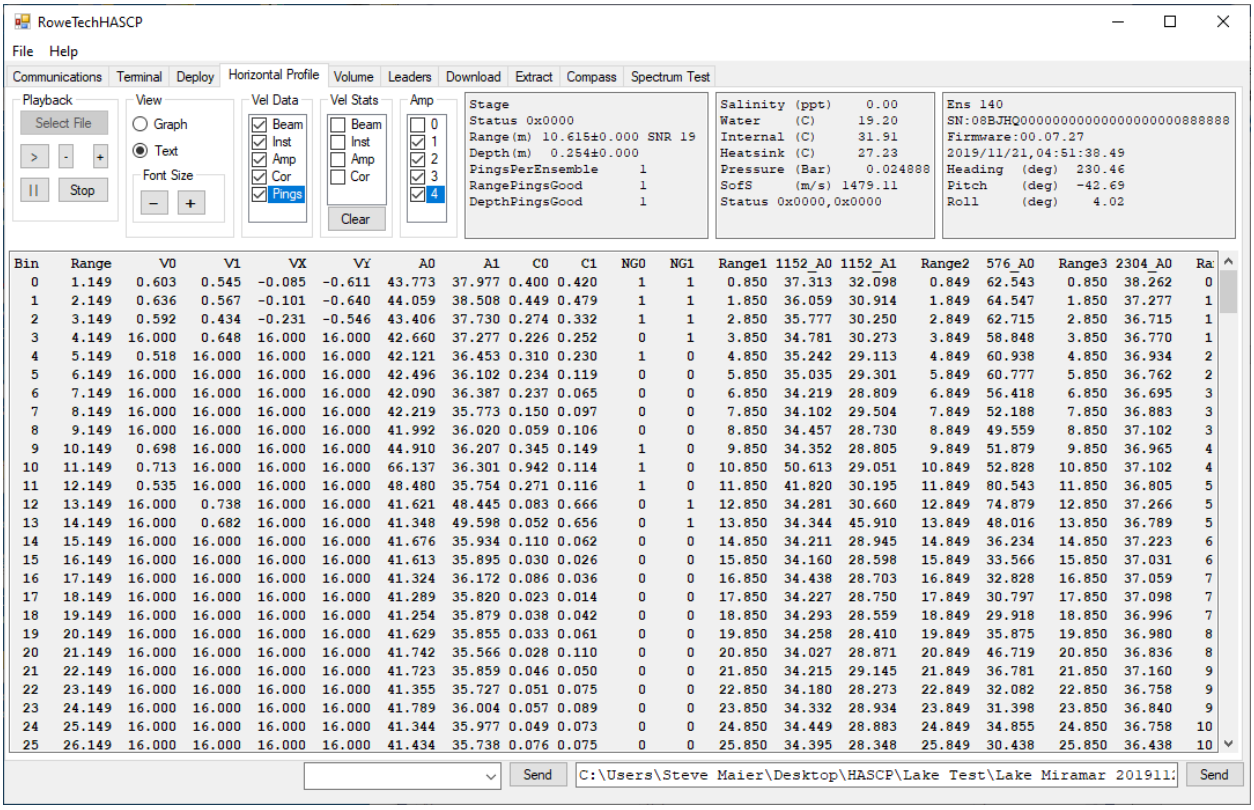
vii. Stage Text Box
    1. Status
        a. 0x0000 indicates no issues
    2. Range (m)
        b. The average range collected during the ensemble.
        c. The standard deviation the range measurements.
        d. The average signal to noise measurement of the surface echo.
    3. Depth (m)
        e. HASCP depth measure by the pressure sensor.
    4. PingsPerEnsemble
        f. The number Pings collected during the ensemble.
    5. RangePingsGood
        g. The number of good SNR Stage pings that were averaged together during the ensemble.
    6. DepthPingsGood
        h. The number of good pressure sensor pings that were averaged together during the ensemble.

viii. Environment and System Status Text Box
    1. Salinity (ppt)
        a. The average salinity value used in the speed of sound calculation.
    2. Water (C)
        b. The averaged (measure or fixed) water temperature used in the speed of sound calculation.
    3. Internal (C)
        c. The average measured temperature on the power regulator circuit board.
    4. Heatsink (C)
        d. The average measured temperature on the transmitter heatsink.
    5. Pressure (BAR)
        e. The average measured pressure.
    6. SofS (m/s)
        f. The average speed of sound (calculated or fixed) used during the ensemble.
    7. Status
        g. 0x0000, 0x0000 indicates no detected issues during the ensemble.

ix. System Info and HPR text Box
    1. Ens
        a. Current ensemble number of data being displayed
    2. SN:
        b. HASCP serial number
    3. Firmware:
        c. Version number
    4. Date Time
        d. Time stamp of first ping in the ensemble.
    5. Heading (deg)
        e. Average Heading collected during the ensemble.
    6. Pitch (deg)
        f. Average Pitch collected during the ensemble.

7. Roll (deg)
    g. Average Roll collected during the ensemble.

x. Horizontal Profile
1. Graph Display (Figure 8)
    a. Graph Corner Values (text boxes near each corner of the graph)
    b. User enterable numbers that scale the graphic display.
2. Text Display (Figure 9)
    a. Data Columns
    b. Numerical data for each selected data.



Bin	Range	V0	V1	VX	VY	A0	A1	C0	C1	NG0	NG1	Range1	1152_A0	1152_A1	Range2	576_A0	Range3	2304_A0	Ra
0	1.149	0.603	0.545	-0.085	-0.611	43.773	37.977	0.400	0.420	1	1	0.850	37.313	32.098	0.849	62.543	0.850	38.262	0
1	2.149	0.636	0.567	-0.101	-0.640	44.059	38.508	0.449	0.479	1	1	1.850	36.059	30.914	1.849	64.547	1.850	37.277	1
2	3.149	0.592	0.434	-0.231	-0.546	43.406	37.730	0.274	0.332	1	1	2.850	35.777	30.250	2.849	62.715	2.850	36.715	1
3	4.149	16.000	0.648	16.000	16.000	42.660	37.277	0.226	0.252	0	1	3.850	34.781	30.273	3.849	58.848	3.850	36.770	1
4	5.149	0.518	16.000	16.000	16.000	42.121	36.453	0.310	0.230	1	0	4.850	35.242	29.113	4.849	60.938	4.850	36.934	2
5	6.149	16.000	16.000	16.000	16.000	42.496	36.102	0.234	0.119	0	0	5.850	35.035	29.301	5.849	60.777	5.850	36.762	2
6	7.149	16.000	16.000	16.000	16.000	42.090	36.387	0.237	0.065	0	0	6.850	34.219	28.809	6.849	56.418	6.850	36.695	3
7	8.149	16.000	16.000	16.000	16.000	42.219	35.773	0.150	0.097	0	0	7.850	34.102	29.504	7.849	52.188	7.850	36.883	3
8	9.149	16.000	16.000	16.000	16.000	41.992	36.020	0.059	0.106	0	0	8.850	34.457	28.730	8.849	49.559	8.850	37.102	3
9	10.149	0.698	16.000	16.000	16.000	44.910	36.207	0.345	0.149	1	0	9.850	34.352	28.805	9.849	51.879	9.850	36.965	4
10	11.149	0.713	16.000	16.000	16.000	66.137	36.301	0.942	0.114	1	0	10.850	50.613	29.051	10.849	52.828	10.850	37.102	4
11	12.149	0.535	16.000	16.000	16.000	48.480	35.754	0.271	0.116	1	0	11.850	41.820	30.195	11.849	80.543	11.850	36.805	5
12	13.149	16.000	0.738	16.000	16.000	41.621	48.445	0.083	0.666	0	1	12.850	34.281	30.660	12.849	74.879	12.850	37.266	5
13	14.149	16.000	0.682	16.000	16.000	41.348	49.598	0.052	0.656	0	1	13.850	34.344	45.910	13.849	48.016	13.850	36.789	5
14	15.149	16.000	16.000	16.000	16.000	41.676	35.934	0.110	0.062	0	0	14.850	34.211	28.945	14.849	36.234	14.850	37.223	6
15	16.149	16.000	16.000	16.000	16.000	41.613	35.895	0.030	0.026	0	0	15.850	34.160	28.598	15.849	33.566	15.850	37.031	6
16	17.149	16.000	16.000	16.000	16.000	41.324	36.172	0.086	0.036	0	0	16.850	34.438	28.703	16.849	32.828	16.850	37.059	7
17	18.149	16.000	16.000	16.000	16.000	41.289	35.820	0.023	0.014	0	0	17.850	34.227	28.750	17.849	30.797	17.850	37.098	7
18	19.149	16.000	16.000	16.000	16.000	41.254	35.879	0.038	0.042	0	0	18.850	34.293	28.559	18.849	29.918	18.850	36.996	7
19	20.149	16.000	16.000	16.000	16.000	41.629	35.855	0.033	0.061	0	0	19.850	34.258	28.410	19.849	35.875	19.850	36.980	8
20	21.149	16.000	16.000	16.000	16.000	41.742	35.566	0.028	0.110	0	0	20.850	34.027	28.871	20.849	46.719	20.850	36.836	8
21	22.149	16.000	16.000	16.000	16.000	41.723	35.859	0.046	0.050	0	0	21.850	34.215	29.145	21.849	36.781	21.850	37.160	9
22	23.149	16.000	16.000	16.000	16.000	41.355	35.727	0.051	0.075	0	0	22.850	34.180	28.273	22.849	32.082	22.850	36.758	9
23	24.149	16.000	16.000	16.000	16.000	41.789	36.004	0.057	0.089	0	0	23.850	34.332	28.934	23.849	31.398	23.850	36.840	9
24	25.149	16.000	16.000	16.000	16.000	41.344	35.977	0.049	0.073	0	0	24.850	34.449	28.883	24.849	34.855	24.850	36.758	10
25	26.149	16.000	16.000	16.000	16.000	41.434	35.738	0.076	0.075	0	0	25.850	34.395	28.348	25.849	30.438	25.850	36.438	10

Figure 9. Horizontal page showing the corresponding measured values of the beams.

3.8 Volume

i. Click the Volume TAB near the top center of the software window. Refer to Figure 10.

ii. Text Display
1. kHz
    a. Beam frequency used in the measurement.
2. Ang(deg)
    a. Beam angle.
3. XmtV
    a. Measure Transmit Voltage
4. Xmt(m)
    a. Transmit length in meters
5. (msec)
    a. Transmit length in time (milliseconds).

6. Pings
    a. Number of pings collected during the ensemble.
7. Good
    a. Number of good pings in the average. Good = meet or exceed the selected signal to noise threshold (Thres).
8. Beg(m)
    a. Start location of the volume average.
9. End(m)
    a. Last location of the volume average. The End location will be adjusted towards the Begin location if the signal to noise level is low.

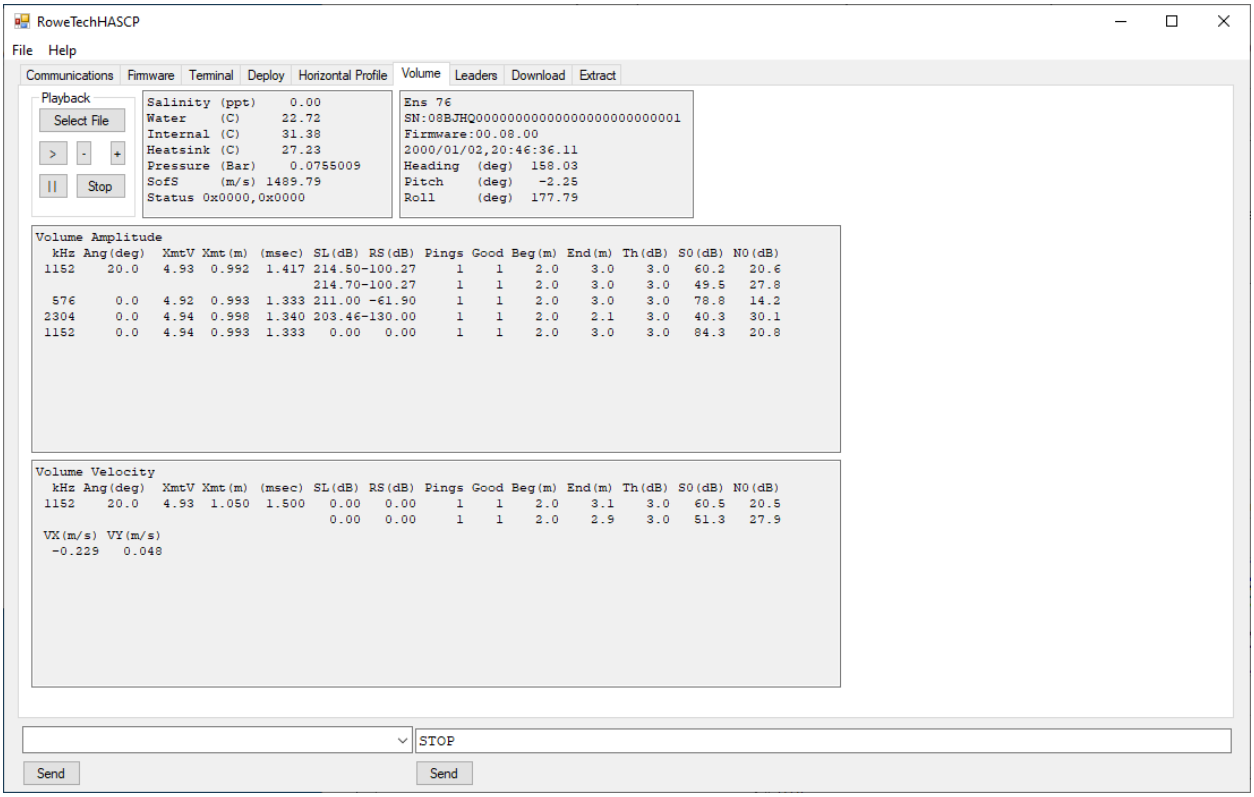


Figure 10. Volume Tab in RTI HASCP software

10. Thresh(dB)
    a. Signal to noise threshold for good data.
11. S0(dB)
    a. Average signal + noise level in the volume measurement.
12. N0(dB)
    a. Average noise level during the measurement.
13. Pings (second beam of Janus pair)
    a. Number of pings collected during the ensemble
14. Good(second beam of Janus pair)
    a. Number of good pings in the average. Good = meet or exceed the selected signal to noise threshold (Thres).
15. Beg(m) (second beam of Janus pair)
    a. Start location of the volume average.
16. End(m) (second beam of Janus pair)

    a. Last location of the volume average. The End location will be adjusted towards the Begin location if the signal to noise level is low.
17. Thresh(dB) (second beam of Janus pair)
    a. Signal to noise threshold for good data.
18. S1(dB) (second beam of Janus pair)
    a. Average signal + noise level in the volume measurement.
19. N1(dB) (second beam of Janus pair)
    a. Average noise level during the measurement.
20. VX(m/s)(Janus pair)
    a. Average X velocity measurement.
21. VY(m/s)(Janus pair)
    a. Average Y velocity measurement.

3.9 Leaders

a. Click the Leaders TAB near the top center of the software window. Refer to Figure 11.
b. Playback
    i. Select File
    ii. > Play
    iii. || Pause
    iv. – Step back one ensemble
    v. + Step forward one ensemble
    vi. Stop
c. RTIHASCP Text Box
    i. Refer to the firmware output data structure description chapter.
d. SYSTEM Text Box
    i. Refer to the firmware output data structure description chapter.
e. JANUS Text Box
    i. Refer to the firmware output data structure description chapter.
f. Amplitude Text Box
    i. Refer to the firmware output data structure description chapter.

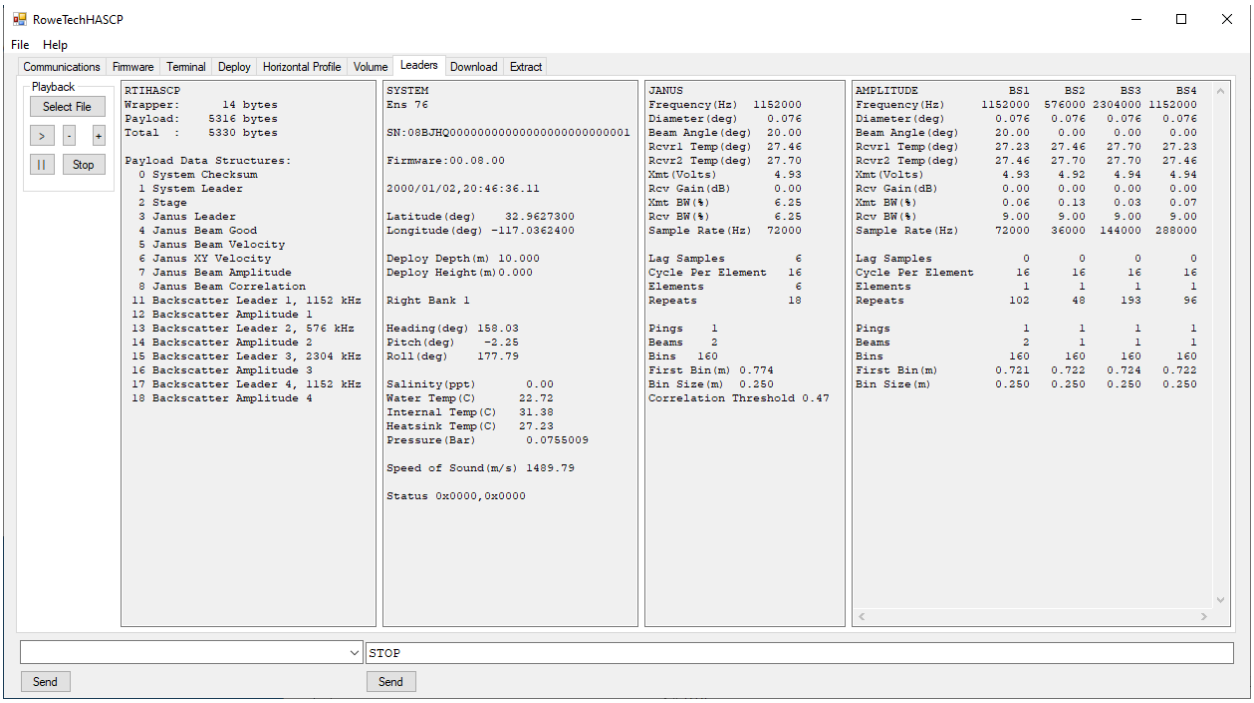


Figure 11. Leaders Tab in RTI HASCP software

3.10 Download

a. Click the Download TAB near the top center of the software window. Refer to Figure 12 and Figure 13.

b. Download Files

ii. Get Directory

1. Clicking Get Directory will send the command DSDIR to the HASCP system via the serial port. The HASCP will respond with a list of all of the files contained on its internal SD card. The software will screen the file names and display only the HASCP data files starting with a "H" and ending with ".bin" (H0000012.bin 2019/11/22 05:15:17 16.005). The directory list is shown in the Upper Text Box. The complete directory list can be viewed on the terminal TAB.

iii. Start Download

1. Clicking Start Download will cause all the "H" files contained on the SD card to be downloaded to the PC.

iv. Cancel Download

1. The download can be Canceled by clicking on the Cancel Download button.

v. First File

1. The name of the first file to download can be entered here. This is useful if some of the data files have previously been downloaded.

vi. sec

1. Displays the elapsed second of the current file being downloaded.

vii. %

1. Displays the percentage of the current file that has been downloaded.

viii. bps

1. Displays the data rate (bit per second) of the current file being downloaded.

ix. Retries
1. Displays the number of times a data packet was lost and re-requested.

x. Tries
1. Displays the number requested data packets.

xi. Bytes
1. Displays the total number of bytes transferred.

c. Top Text Box

xii. Shows Directory data and status during download.

d. Bottom Text Box

xiii. Shows download file storage location and file status.

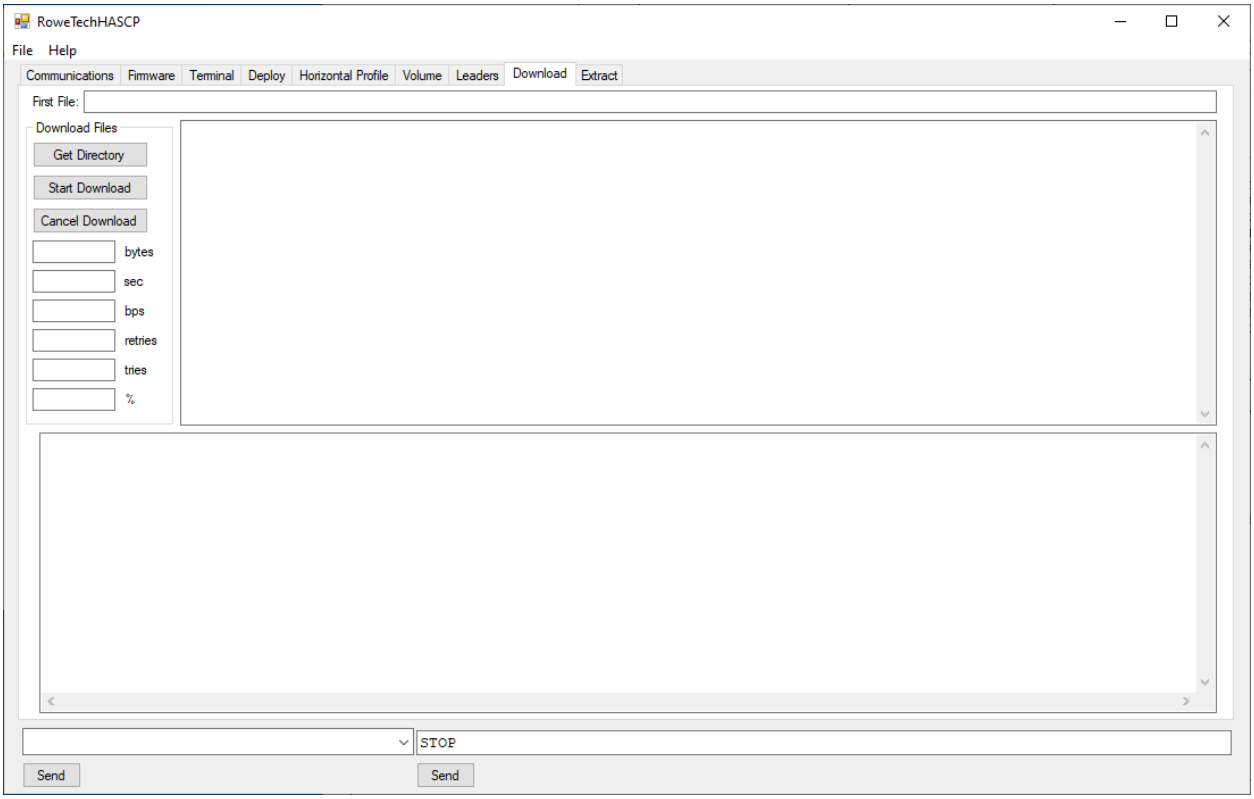


Figure 12. Download Tab in RTI HASCP software with no files

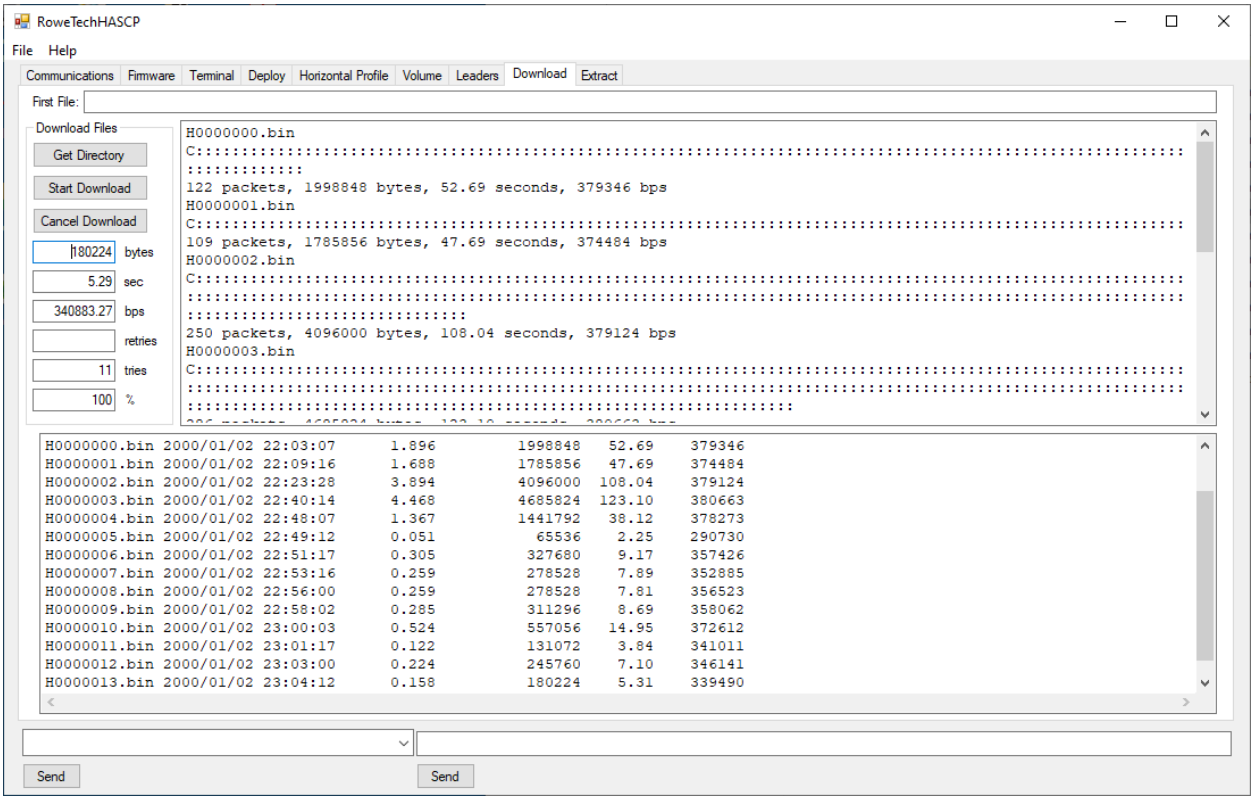


Figure 13. Download Tab in RTI HASCP software showing files getting downloaded.

3.11 Extract

a. Click the Extract TAB near the top right-center of the software window. Refer to Figure 14.
b. Binary to CSV
    i. Clicking the Binary to CVS button allows the user to select a binary HASCP file stored on the PC which is then converted to multiple Comma Separated Variable (CSV) files.
c. Ensemble Number
    i. Display the current ensemble number being converted and save on the PC.
d. Status Text Box
    i. Displays a list of the CSV files begin saved and their location on the PC.
    ii. “Extract Complete” will be displayed when the conversion is complete.

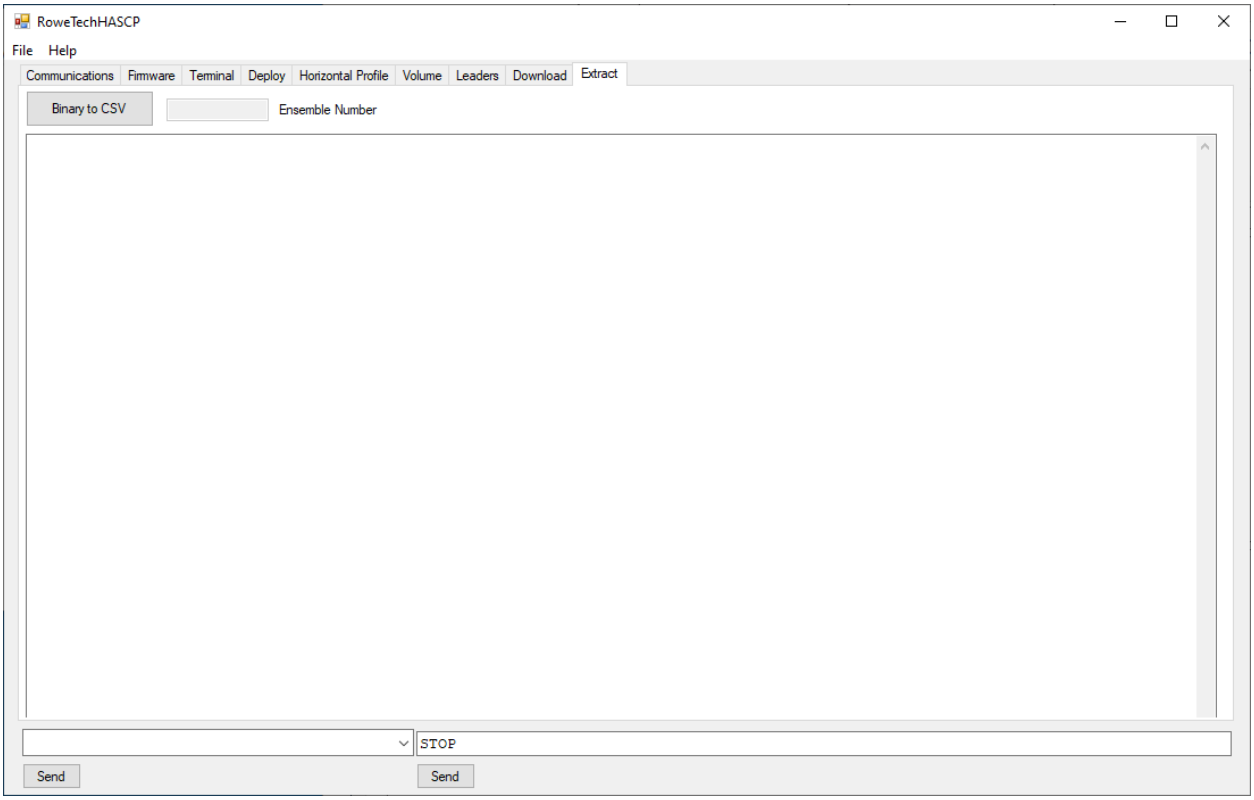


Figure 14. Extract tab in the RTI HASCP software

3.12 Firmware upload

a. Open the Firmware Tab as shown in Figure 15.
b. Click the Current version to check to see the current firmware version.
c. Contact RTI if a new firmware upgrade is required.
d. Click the update button to select the new firmware to be updated. The firmware update process will update the process as it is getting updated and display the new firmware version once the process is completed.

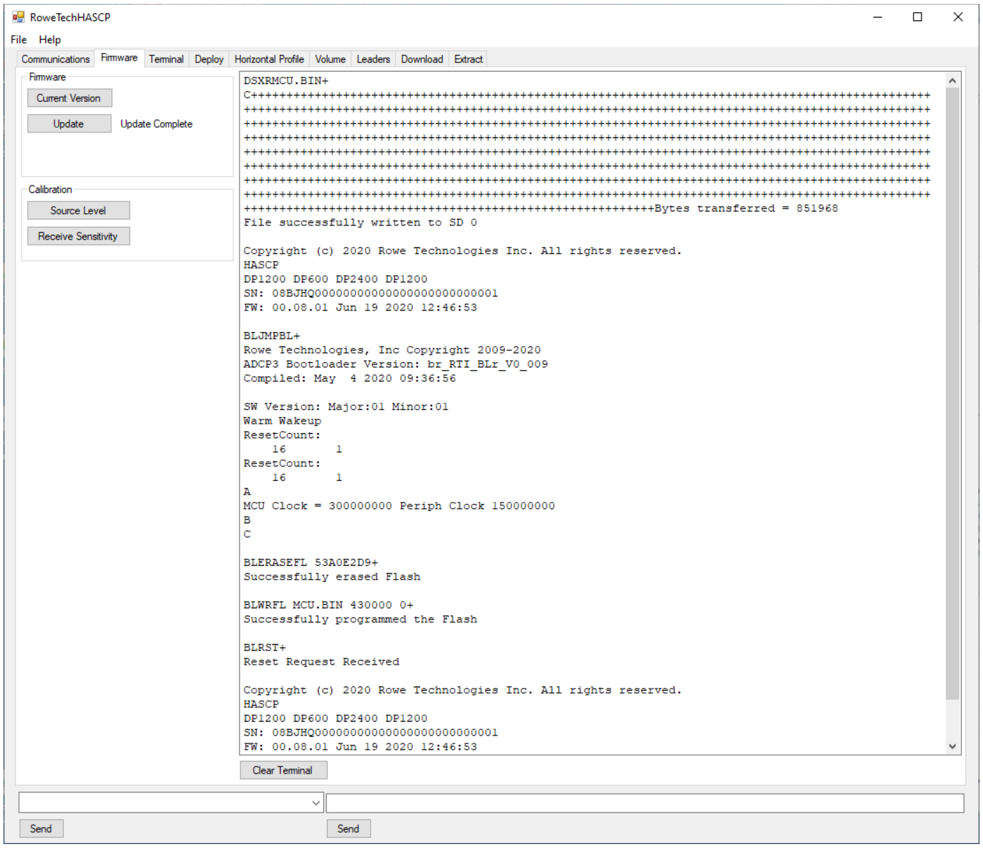


Figure 15. Firmware Tab in the RTI HASCP software

4 Connection Diagnostics

This procedure presumes the operator has basic knowledge of the ROWETECH HASCP and has experience with serial communication interfaces and test equipment. For further assistance, contact Rowe Technologies directly.

Connect HASCP communication port(s) to the Windows PC (outlined above). Open the ROWETECH HASCP software. Select communication port and Baud rate.

Select the Terminal screen on the HASCP software and turn on the power supply to the HASCP and click Break <CR>.

If you see the Wake Up message on the Terminal screen:

HASCP
DP1200 DP600 DP2400 DP1200
SN: 08BJHQ00000000000000000000888888
FW: 00.07.27 Nov 20 2019 05:22:14
STOP+

If there is no Wakeup message displayed on the Terminal screen:

- Verify the Baud rate and Communication port setting.
- Verify the wiring of the communication port.
- Verify RS232 communications on the PC by connecting pins 2 and 3 together on the DB9 connector going into the PC. Then send a command for e.g. START to the port. You should see the command/characters echoed on the Terminal screen.
- Try a NULL MODEM on the RS232.
- Connect an Oscilloscope to the RS485 data lines. Then send a command for e.g. START. You should see the command/characters on the scope display the bit rate should match the PC Baud rate.
- Try swapping the 2 data lines on the RS485.
- Connect an oscilloscope to the ADCP/DVL output data lines on either communication port.
- Turn OFF the HASCP power. Wait 10 seconds. Turn ON the HASCP power supply:

Measure the voltage at the input to the HASCP pin1 on the power connector. The voltage should be between 12 and 24 VDC depending on the power supply. If the voltage is out of range find another supply.

Measure the power supply current. The power up current should be 0.2 to 0.5 Amps if the system is awake and waiting for a command. If the system is pinging it will be drawing 2 to 3 Amps and 0.2 to 0.3 Amps the rest of the time.

Remove the underwater housing keeping the wiring connectors attached. If there is still an issue contact ROWETECH with the results of these tests.

5 Preparing for a Deployment

RTI recommends the following checklist that may help the user towards a successful deployment.

Structural Integrity – The structural integrity of cables and connectors are also important, be sure to there are no cuts or cracks in the cable or connectors. It is important to make sure that when reassembling the system that the nuts and bolts are tightened sufficiently. In addition, It is essential that all O-rings are properly greased and seated in the groove found in the transducer head.

Cable Connections – Be certain that all the cables are properly connected as outlined in Section 2. When using a wet-mateable connection, be sure to first mate the connectors completely and then tighten the locking sleeve. This is the proper procedure for wet-mateable connections (Do not connect the wet-mateable connectors partially and then use the locking sleeve to tighten – this will not provide an adequate seal for the connector).

Power – Make sure that the power connections are properly connected.

6 Instrument Care

Below are some general guidelines to taking care of HASCP.

6.1 Guidelines to Instrument Care

Please consider the following guidelines to prolong the life of your instrument, decrease the risk of damage and continue its factory tested performance:

Please do not open the instrument housing enclosure unless you have contacted service at ROWETECH. There is no regular field maintenance to perform on these units.

Handle with care- dropping the unit or severe impact could damage the transducer elements, the water tight integrity of the instrument housing and the internal electronics of the system.

Avoid leaving the instrument in direct sun light for long periods of time. If they do need to be placed in direct sun light consider a cover. Try not to store or transport these items in extreme temperatures. When the instrument and deck box are not in use please place them back in the original shipping container.

Keep the instrument clean and clear of dirt, oils and any chemicals. Dirt may contaminate the O-ring seals and the electrical connection of the underwater connector.

The transducer face (Red color) is manufactured with a special urethane. The urethane is also softer than the other exterior acetal housing material. The transducer faces must be given extra care to avoid punctures, cuts, direct sun light, any chemical contact, discoloration from oils and contaminates in the water. The transducer face is soft enough to sustain impressions from anything left on it or if it is placed against or on top of anything.

The instrument has an internal magnetic flux gate compass. The compass may not work correctly if the instrument is near a steel structure or magnetic field.

Please do not “ping” your instrument in air for any prolong amounts of time, this could cause damage to the transducer and the electronics. Please immerse the unit in a container of water for lab testing.

7 Firmware Details

7.1 System files stored on the SD card

A. bbcode.bin
    a. Contains broadband transmit codes
B. helpsed.txt
    a. Command help file which is output when Help<CR> is sent to the system.
C. SYSCONF.BIN
    a. Stored command parameters. Created when CSAVE<CR> is sent to the system.
D. sleep.bin
    a. When the system is sleeping the ping accumulators are stored here.
E. HFILE.TXT
    a. Keeps track of the current file number for storing the binary ensemble data.
F. H0000001.bin
    a. Binary ensemble data file. The number after the H is incremented when the file size exceeds 16 Mbytes.

7.2 Sub System Codes

HASCP system Serial Numbers begin with 08. The Subsystem Codes which make up the next several digits in the serial number are listed below.

Typical HASCP SN: 08BJHQ00000000000000000000123456

Code	Description
0	Reserved
A	2.4 MHz 2 beam 20 degree piston
B	1.2 MHz 2 beam 20 degree piston
C	600 kHz 2 beam 20 degree piston
D	300 kHz 2 beam 20 degree piston
E	150 kHz 2 beam 20 degree piston
F	Spare
G	Spare
H	2.4 MHz 0 degree piston
I	1.2 MHz 0 degree piston
J	600 kHz 0 degree piston
K	300 kHz 0 degree piston
L	150 kHz 0 degree piston
M	Spare
N	Spare
O	Spare
P	2.4 MHz upward facing piston
Q	1.2 MHz upward facing piston
R	600 kHz upward facing piston
S	300 kHz upward facing piston
T	150 kHz upward facing piston
S	Spare

U Spare

7.3 HASCP Commands

7.3.a Command Summary

The Tables below lists the various commands used by the HASCP software.

Table 4. System Deployment Commands

Command	Description
START	Start ADCP pinging. *Note*: Once started, the system won't respond to any commands except STOP, STIME and CSHOW commands.
STOP	Stop pinging and return to the command input mode.
SLEEP	Power down system.
SLEEPA	Set wakeup alarm for 10 seconds then Power down system.
SLEEPSECONDS i	Seconds to wait before going to sleep when system is in the STOP state.
SPOS	Display System Geo Position.
SPOS i, ii, iii, iv, v	Enter the system Geo Position, in which i. Latitude (deg) ii. Longitude (deg) iii. Deployment Depth (m) iv. Height Above Bottom (m) v. River Bank 0 = Left Bank, 1 = Right Bank
STIME	Display the System Time.
STIME YYYY/MM/DD,HH:MM:SS	Set the System Time.

Table 5. Firmware Commands

Command	Description
FMSHOW	Show the Firmware Version

Table 6. Data Storage Commands

Command	Description

DSXRfilename.abc	XMODEM file upload. This command is used to transfer a file, via the serial communication link, from an external device to the SD card contained within the ROWETECH system. File names are limited to a maximum of 8 characters before the extension.
DSXTfilename.abc	XMODEM file download. This command is used to transfer a file, via the serial communication link, from the SD card contained within the ROWETECH system to an external device. File names are limited to a maximum of 8 characters before the extension.
DSFORMAT	Format / Erase the SD card.
DSDIR	Show SD card directory of stored files.
DSSHOW	Show SD card capacity and usage.

Table 7 Configuration Commands

Command	Description
CLOAD	Load the configuration file "SYSCONF.BIN" from SD card to the current command set.
CSAVE	Save the current command set to the configuration file "SYSCONF.BIN" on SD card.
CSHOW	Show command set.
CDEFAULT	Restores the system configuration to factory defaults. *Note: Actual default values will vary depending on system type.*

Water Profile Commands

Note: Water Profiling commands can be followed by a pair of brackets and a number (i.e. CWPON[1] 1 <CR>) to set a selected subsystem configuration. The brackets are optional for subsystem zero [0].

Table 8 Water Profile Commands

Command	Description
CWPON i	Enable Type of Processing i 0 = disable 1 = velocity 2 = sediment 3 = stage
CWPBB i, ii	Water Profile Broad Band. Sets water profile coded pulse transmission and lag. i. Acoustic Transmit Type • 0 = Narrowband • 1 = Broadband Coded Pulse • 2 = Broadband Non-Coded Pulse

	ii. Broadband Transmit Lag in vertical meters
CWPBL i	Water Profile Blank in vertical meters.
CWPBS i	Water Profile Bin Size in vertical meters.
CWPBN i	Water Profile Bins.
CWPP i	Pings Per Ensemble.
CWPTBP i	Time Between Pings.
CWPMS i	Horizontal Max Speed (m/s), Not implemented.
CWPRC i, ii, iii, iv, v, vi, vii, viii	Receiver Hardware Control i. Phased Array Beamformer • 0 = Disabled • 1 = Enabled ii. Bandpass Filter Bandwidth • 0 = Width • 1 = Narrow iii. Phased Array Vertical Beam • 0 = Off • 1 = On iv. Receiver Built in Test • 0 = Off • 1 = On v. Lowpass Filter Bandwidth • 0 = Width • 1 = Narrow vi. Spare vii. Receiver Power Control • 0 = OFF • 1 = ON viii. Preamp Gain • 0 = Low Gain • 1 = High Gain
CWPRF i, ii, iii	Internal Power Supply Switching Frequency i. F2 ii. F1 iii. F0
CWPRT i, ii, iii, iv	Water Profile Range Tracking i. Water Profile Range Tracking Mode ii. Begin bin if m = 1 or % of pressure depth if m = 2 iii. End bin iv. SNR (dB) Threshold
CWPST i, ii, iii	Velocity Screening Threshold i. Correlation Threshold (0 to 1) ii. Q Good Threshold (m/s)

	iii. V Good Threshold (m/s)
CWPX i, ii	Water Profile Transmit i. Transmit Length (m), if set to 0 transmit will set equal to bin length ii. Broadband Code Selection • 0 = full length code • 1 = single element code (reduced transmit power)
CWPTC i, ii, iii	Transmit Control i. Beam Transmit Enable • 000001111 = beams 3 to 0 enabled • 011110000 = beams 7 to 4 enabled • 100000000 = beam 8 enabled ii. Broadband Transmit Bandwidth • 0=default, 1=50%, 2=25%, 3=12.5%,4=6.25%,5=3.125%,6=1.5625% iii. Broadband Sample Rate • 0=default, 1=50%, 2=25%, 3=12.5%,4=6.25%,5=3.125%,6=1.5625%
CWPTP i	i. Transmit Power (binary) • 1000 High Power • 0100 • 0011 Mid Power • 0001 • 0000 Low Power

Table 9. ADCP Ensemble Commands

Command	Description
CEAUTOSTART i	Auto start pinging i. 0 = Disable Auto Start ii. 1 = Enable Auto Start on power up
CEI HH:MM:SS.hh	Set the Ensemble Interval
CETFP YYYY/MM/DD,HH:mm:SS.hh	Set the Time of First Ping
CERECORD i, ii, iii	i. Ensemble Recording • 0 = Disable • 1 = Enable ii. Single Ping Recording • 0 = Disable • 1 = Enable iii. Data Logger Trigger • 0 = Disable • 1 = Enable

C232OUT i	Serial Data Output on the RS232 port • 0 = Disable • 6 = Enable
C485OUT i	Serial Data Output on the RS485 port • 0 = Disable • 6 = Enable
C422OUT i	Use C232OUT to control RS422 output

Table 10. Environmental Commands

Command	Description
CEAUTOSTART i	Auto start pinging iii. 0 = Disable Auto Start iv. 1 = Enable Auto Start on power up
CEI HH:MM:SS.hh	Set the Ensemble Interval
CETFP YYYY/MM/DD,HH:mm:SS.hh	Set the Time of First Ping
CWSSC i, ii, iii, iv	Water Speed of Sound Control i. Water Temperature • 0 = Command • 1 = Sensor ii. Transducer Depth • 0 = Command • 1 = Sensor iii. Salinity • 0 = Command • 1 = Sensor iv. Speed of Sound • 0 = Command • 1 = Sensor • 2 = Internal Calculation
CWS i	i. Water Salinity (ppt)
CWT i	i. Water Temperature (degree)
CTD i	i. Transducer Depth (m)
CWSS i	

	i. Water Speed of Sound (m/s)
CHO i, ii, iii	i. Heading Offset (±180 deg) ii. System to Ship Offset (±180 deg) iii. System to PNI Offset (±180 deg)
CHS i	i. Heading Source • 0 = None • 1 = PNI • 2 = $HDT (NMEA)
CTS i	i. Tilt Source • 0 = None • 1 = PNI
CVSF i, ii	i. Water Velocity Scale Factor ii. Bottom Velocity Scale Factor
CPZ	Zero Pressure Sensor

Table 11. Coordinate Systems Commands

Command	Description
XFRMBEAMSHOW	Show the 4x4 beam direction matrix
XFRMBEAMSAVE 0,1,2,...15	Read in and save a 4x4 beam direction matrix
XFRMBEAMSAVED	Set the beam direction matrix to the default values
XFRMBEAMSAVEE	Erase the saved beam direction matrix

Table 12. Communications Commands

Command	Description
C232B i, ii, iii, iv	RS232 Serial Data Control
C485B i, ii, iii, iv	RS485 Serial Data Control
C422B i, ii, iii, iv	RS422 Serial Data Control i. Baud Rate (bits per second): 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400,460800, 921600. ii. Number of data bits 7 or 8. iii. Parity

	• 0 = None • 1 = Odd • 2 = Even iv. Stop Bits: 1 = One, 2 = Two
C485G i	i. Seconds for RS485 Gap
CTRIG i	i. External Trigger Input • 0 = Disable • 1 = High State • 2 = Low State • 3 = Low to High • 4 = High to Low

Table 13 Diagnostics Commands

Command	Description
DIAGCPT	Enable Compass Pass Thru
DIAGPRESSURE	Show Pressure Sensor information
DIAGSD	Show the SD card register values and directory
DIAGPNI i	Check PNI Compass • 0 = Show PNI Information • 1 = Show Heading, Pitch, Roll • 2 = Diagnostic Ping Continuously
DIAGSAMP i	i. Ping the System Hardware • 1 = Diagnostic Ping • 2 = Diagnostic Ping continuously
DIAGSPECTRUM i	i. Number of samples for each point (default 2000)
DIAGRTC i	i. 1 = Show RTC registers ii. 2 = Show RTC registers continuously

Table 14. Environmental Commands

Command	Description

CWSSC *a, b, c, d*	Sets where the source for speed of sound calculation: 0 = Command, 1 = Sensor, 2 = Internal Calculation. Parameters: *a*: Water Temperature Source Typically, *a* = 1 uses the build-in temperature sensor. Or *a* = 0 from CWT command. *b*: Transducer Depth Source Typically, *b* = 1 from pressure sensor. Or *b* = 0, from CTD command. *c*: Salinity Source Typically, *c* = 0, uses CWS command to set salinity. *d*: Speed of Sound Source Typically, *d* = 2 internal calculation.
CWS *n.nn*	Water Salinity (ppt). Typically, 35 for salt water and 0 for fresh water.
CWT *n.nn*	Water Temperature (degrees).
CTD *n.nn*	Transducer Depth (meters).
CWSS *n.nn*	Water Speed of Sound (m/s).
CHO *n.n, m.m, o.o*	Heading Offset. Parameters *n.n*: Heading offset (-180 to + 180) added to the compass or GPS heading in the system. *m.m*: System to ship heading offset (-180 to +180). *o.o*: System to PNI compass offset (-180 to +180).
CHS *n*	Heading Source. Selects the heading source for ENU transformations. Parameters: n = 0: no heading n = 1: internal (PNI) compass n= 2: GPS HDT via serial port
CTS *n*	Tilt Source. Selects the tilt source for ENU transformations. n = 0: no tilt, n = 1: internal (PNI) compass.
CVSF *n.nn,m.mm*	Velocity Scale Factor. Scale factor *n.nn* is applied to all water velocity measurement data. Scale factor *m.mm* is applied to all bottom velocity measurement data. Parameters: *n.nn*: Water Velocity Scale Factor. New Velocity = CVSF *n.nn* * Velocity *m.mm*: Bottom Track Scale Factor. New Velocity = CVSF *m.mm* * Velocity
CPZ	Sets the current pressure reading to the zero point (if pressure sensor is installed).

7.4 Data Structure ID list

These codes are used to identify the data structure types contained in the Output Data.

0. SYSTEM_CHECKSUM
1. SYSTEM_LEADER
2. VERTICAL_STAGE
3. HORIZONTAL_JANUS_LEADER
4. HORIZONTAL_JANUS_GOOD
5. HORIZONTAL_JANUS_BEAM
6. HORIZONTAL_JANUS_INSTRUMENT
7. HORIZONTAL_JANUS_AMPLITUDE
8. HORIZONTAL_JANUS_CORRELATION
9. BACKSCATTER_LEADER_00
10. BACKSCATTER_PROFILE_00
11. BACKSCATTER_LEADER_01
12. BACKSCATTER_PROFILE_01
13. BACKSCATTER_LEADER_02
14. BACKSCATTER_PROFILE_02
15. BACKSCATTER_LEADER_03
16. BACKSCATTER_PROFILE_03
17. BACKSCATTER_LEADER_04
18. BACKSCATTER_PROFILE_04

7.5 Ensemble Output Data Structure (little endian)

Data structures contained in the output data are stored as a linked list. Each data structure has 4 bytes at the beginning containing the ID and Size (bytes in the structure). The structure data begins at the next byte after the Size. To find the ID address of the next linked structure add the current Size to the next address after the current Size.
Each ensemble has a wrapper and a payload. The wrapper has an 8 byte header to help identify the start of ensemble. After the header the payload size, in bytes, is used to locate the end of the ensemble where the checksum is stored. An inverted payload size is included to help check data integrity.

The payload checksum is calculated using code shown below.

```
unsigned short rtiprocA_CRCcalc(void * data, int len)
{
    unsigned char * dataPtr = (unsigned char *)data;
 int index = 0;
 unsigned short crc = 0;
 while(len--)
 {
    crc = (unsigned char)(crc >> 8) | (crc << 8);
    crc ^= dataPtr[index++];
    crc ^= (unsigned char)(crc & 0xff) >> 4;
    crc ^= (crc << 8) << 4;
    crc ^= ((crc & 0xff) << 4) << 1;
 }
 return crc;
}
```

If the ensemble checksum matches the calculated check sum and the inverted payload size matches the payload size inverted, the ensemble data is considered to be valid.

//**Payload Wrapper**
byte Header[8];// 52 54 49 48 41 53 43 50 (bytes 0 thru 7)
ushort PayloadSize ;//LSB, MSB (bytes 8,9)
ushort Inverted;//LSB, MSB (bytes 10,11)

//**SYSTEM_LEADER**
ushort System_ID;// 0,1
ushort System_Bytes;// 2,3
ulong System_Ensemble_Number;//LSB, , ,MSB 4 thru 7
byte System_Serial_Number[32];// 8 thru 39
ushort System_Firmware_MAJOR;//40 41
ushort System_Firmware_MINOR;//42 43
ushort System_Firmware_REVISION;//44 45
ushort System_year;//46 47
byte System_month;//48
byte System_day;//49
byte System_hour;//50
byte System_minute;//51
byte System_second;//52
byte System_hsec;//53
double System_Latitude;//54 thru 61
double System_Longitude;//62 thru 69
single System_Water_Depth;//70 thru 73
single System_Pressure_Sensor_Height;//74 thru 77
ushort System_Right_Bank//78 79
ushort System_HeadingX100;//80 81
ushort System_PitchX100;//82 83
ushort System_RollX100;//84 85
ushort System_SalinityX100;//86 87
ushort System_WaterTemperatureX100;//88 89
ushort System_BackPlaneTemperatureX100;//90 91
ushort System_HeatSink1TemperatureX100;//92 93
single System_Pressure;//94 thru 97
single System_SpeedOfSound;//98 thru 101
ushort System_Status;//102 103
ushort System_Status2;//104 105

//If Stage is enabled CWPON[4] 3<CR>
// **VERTICAL_STAGE**
ushort Stage_ID; //0 1
ushort Stage_Bytes;// 2,3
ushort Stage_PingCount;//4 5
single Stage_Range;//6 thru 9
single Stage_Range_std;//10 thru 13
single Stage_Range_SNR;//14 thru 17
ushort Stage_Range_Pings_Good;//18 19
single Stage_Pressure_Depth;//20 thru 23
single Stage_Pressure_Depth_std;//24 thru 27
ushort Stage_Pressure_Depth_Pings_Good;//28 29
ushort Stage_Status;//30 31 Status

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_LEADER**
ushort Janus_ID; // 0 1
ushort Janus_Bytes;// 2,3
single Janus_SystemFreqHz[sub];//4 thru 7
ushort Janus_Transducer_DiameterX1000;//8 9
ushort Janus_BeamAngleDegreesX1000;//10 11
ushort Janus_Rcvr1TemperatureX100;//12 13
ushort Janus_Rcvr2TemperatureX100;//14 15
ushort Janus_TransmitVoltsX100;//16 17
ushort Janus_PreAmpGainX1000;//18 19
ushort Janus_TransmitBwX65535;//20 21 (1/T or 1/CPE) * 65535
ushort Janus_ReceiveBwX65535);//22 23
single Janus_SamplesPerSecond;//24 thru 27
ushort Janus_LagSamples;//28 29
ushort Janus_CPCE;//30 31
ushort Janus_NCE;//32 33
ushort Janus_RepeatN;//34 35
ushort Janus_PingCount;//36 37
ushort Janus_Beams;//38 39
ushort Janus_Bins;//40 41
ushort Janus_FirstBinDepthX1000;//42 43
ushort Janus_BinSizeX1000;//44 45
ushort Janus_CorrelationThresholdX1000; //46 47
ushort Janus_Vol0BeginX1000;// 48 49
ushort Janus_Vol1BeginX1000;//50 51
ushort Janus_Vol0EndX1000;//52 53
ushort Janus_ Vol1EndX1000;//54 55
ushort Janus_Vol0AmpX100;//56 57
ushort Janus_Vol1AmpX100;//58 59
ushort Janus_Vol0VelX1000;// 60 61
ushort Janus_Vol1VelX1000;//62 63
ushort Janus_Vol0VelXx1000;//64 65
ushort Janus_Vol1VelYx1000;//66 67
ushort Janus_VolThresholdX100;//68 69
ushort Janus_VolGoodPings0;//70 71
ushort Janus_VolGoodPings1;//72 73

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_GOOD**
ushort Janus_Good_ID; // 0 1
ushort Janus_Good_Bytes;// 2,3
ushort Janus_Good_N[beams][bins];// 4 + 2 * beams * bins

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_BEAM**
ushort Janus_Beam_ID; // 0 1
ushort Janus_Beam_Bytes;// 2,3
ushort Janus_Beam_VelX2000[2][bins];// 4 + 2 * beams * bins

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_INSTRUMENT**
ushort Janus_Instrument_ID; // 0 1
ushort Janus_Instrument_Bytes;// 2,3
ushort Janus_Instrument_VelX2000[2][bins];// 4 + 2 * beams * bins

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_AMPLITUDE**
ushort Janus_Amplitude_ID; // 0 1
ushort Janus_Amplitude_Bytes;// 2,3
ushort Janus_AmplitudeX256[2][bins];// 4 + 2 * beams * bins

//If Velocity Profile is enabled CWPON[0] 1<CR>
//**HORIZONTAL_JANUS_CORRELATION**
ushort Janus_Correlation_ID; // 0 1
ushort Janus_Correlation_Bytes;// 2,3
ushort Janus_CorrelationX16384[2][bins];// 4 + 2 * beams * bins

BACKSCATTER_LEADER_00;// Not used

//If Backscatter is enabled CWPON[1] 2<CR>
// **BACKSCATTER_LEADER_01**
ushort BS1_ID; // 0 1
ushort BS1_Bytes;// 2,3
single BS1_SystemFreqHz[sub];//4 thru 7
ushort BS1_Transducer_DiameterX1000;//8 9
ushort BS1_BeamAngleDegreesX1000;//10 11
ushort BS1_Rcvr1TemperatureX100;//12 13
ushort BS1_Rcvr2TemperatureX100;//14 15
ushort BS1_TransmitVoltsX100;//16 17
ushort BS1_PreAmpGainX1000;//18 19
ushort BS1_TransmitBwX65535;//20 21 (1/T or 1/CPE) * 65535
ushort BS1_ReceiveBwX65535);//22 23
single BS1_SamplesPerSecond;//24 thru 27
ushort BS1_LagSamples;//28 29
ushort BS1_CPCE;//30 31
ushort BS1_NCE;//32 33
ushort BS1_RepeatN;//34 35
ushort BS1_PingCount;//36 37
ushort BS1_Beams;//38 39
ushort BS1_Bins;//40 41
ushort BS1_FirstBinDepthX1000;//42 43
ushort BS1_BinSizeX1000;//44 45
ushort BS1_Vol0BeginX1000;//46 47
ushort BS1_Vol1BeginX1000;//48 49
ushort BS1_Vol0EndX1000;//50 51
ushort BS1_ Vol1EndX1000;//52 53
ushort BS1_Vol0AmpX100;//54 55
ushort BS1_Vol1AmpX100;//56 57
ushort BS1_Vol0VelX1000;//58 59
ushort BS1_Vol1VelX1000;//60 61

ushort BS1_Vol0VelYx1000;//62 63
ushort BS1_Vol1VelYx1000;//64 65
ushort BS1_VolThresholdX100;//66 67
ushort BS1_VolGoodPings0;//68 69
ushort BS1_VolGoodPings1;//70 71

//If Backscatter is enabled CWPON[1] 2<CR>
// **BACKSCATTER_PROFILE_01**
ushort BS1_Amplitude_ID; // 0 1
ushort BS1_Amplitude_Bytes;// 2,3
ushort BS1_AmplitudeX256[2][bins];// 4 + 2 * beams * bins

//If Backscatter is enabled CWPON[2] 2<CR>
// **BACKSCATTER_LEADER_02**
ushort BS2_ID; // 0 1
ushort BS2_Bytes;// 2,3
single BS2_SystemFreqHz[sub];//4 thru 7
ushort BS2_Transducer_DiameterX1000;//8 9
ushort BS2_BeamAngleDegreesX1000;//10 11
ushort BS2_Rcvr1TemperatureX100;//12 13
ushort BS2_Rcvr2TemperatureX100;//14 15
ushort BS2_TransmitVoltsX100;//16 17
ushort BS2_PreAmpGainX1000;//18 19
ushort BS2_TransmitBwX65535;//20 21 (1/T or 1/CPE) * 65535
ushort BS2_ReceiveBwX65535);//22 23
single BS2_SamplesPerSecond;//24 thru 27
ushort BS2_LagSamples;//28 29
ushort BS2_CPCE;//30 31
ushort BS2_NCE;//32 33
ushort BS2_RepeatN;//34 35
ushort BS2_PingCount;//36 37
ushort BS2_Beams;//38 39
ushort BS2_Bins;//40 41
ushort BS2_FirstBinDepthX1000;//42 43
ushort BS2_BinSizeX1000;//44 45
ushort BS2_VolBeginX1000;//46 47
ushort BS2_VolEndX1000;//48 49
ushort BS2_VolAmpX100;//50 51
ushort BS2_VolThresholdX100;//52 53
ushort BS2_VolGoodPings;//54 55

//If Backscatter is enabled CWPON[2] 2<CR>
// **BACKSCATTER_PROFILE_02**
ushort BS2_Amplitude_ID; // 0 1
ushort BS2_Amplitude_Bytes;// 2,3
ushort BS2_AmplitudeX256[bins];// 4 + 2 * bins

//If Backscatter is enabled CWPON[3] 2<CR>
// **BACKSCATTER_LEADER_03**
ushort BS3_ID; // 0 1
ushort BS3_Bytes;// 2,3

single BS3_SystemFreqHz[sub];//4 thru 7
ushort BS3_Transducer_DiameterX1000;//8 9
ushort BS3_BeamAngleDegreesX1000;//10 11
ushort BS3_Rcvr1TemperatureX100;//12 13
ushort BS3_Rcvr2TemperatureX100;//14 15
ushort BS3_TransmitVoltsX100;//16 17
ushort BS3_PreAmpGainX1000;//18 19
ushort BS3_TransmitBwX65535;//20 21 (1/T or 1/CPE) * 65535
ushort BS3_ReceiveBwX65535);//22 23
single BS3_SamplesPerSecond;//24 thru 27
ushort BS3_LagSamples;//28 29
ushort BS3_CPCE;//30 31
ushort BS3_NCE;//32 33
ushort BS3_RepeatN;//34 35
ushort BS3_PingCount;//36 37
ushort BS3_Beams;//38 39
ushort BS3_Bins;//40 41
ushort BS3_FirstBinDepthX1000;//42 43
ushort BS3_BinSizeX1000;//44 45
ushort BS3_VolBeginX1000;//46 47
ushort BS3_VolEndX1000;//48 49
ushort BS3_VolAmpX100;//50 51
ushort BS3_VolThresholdX100;//52 53
ushort BS3_VolGoodPings;//54 55

//If Backscatter is enabled CWPON[3] 2<CR>
// **BACKSCATTER_PROFILE_03**
ushort BS3_Amplitude_ID; // 0 1
ushort BS3_Amplitude_Bytes;// 2,3
ushort BS3_AmplitudeX256[bins];// 4 + 2 * bins

//If Backscatter is enabled CWPON[4] 2<CR>
// **BACKSCATTER_LEADER_04**
ushort BS4_ID; // 0 1
ushort BS4_Bytes;// 2,3
single BS4_SystemFreqHz[sub];//4 thru 7
ushort BS4_Transducer_DiameterX1000;//8 9
ushort BS4_BeamAngleDegreesX1000;//10 11
ushort BS4_Rcvr1TemperatureX100;//12 13
ushort BS4_Rcvr2TemperatureX100;//14 15
ushort BS4_TransmitVoltsX100;//16 17
ushort BS4_PreAmpGainX1000;//18 19
ushort BS4_TransmitBwX65535;//20 21 (1/T or 1/CPE) * 65535
ushort BS4_ReceiveBwX65535);//22 23
single BS4_SamplesPerSecond;//24 thru 27
ushort BS4_LagSamples;//28 29
ushort BS4_CPCE;//30 31
ushort BS4_NCE;//32 33
ushort BS4_RepeatN;//34 35
ushort BS4_PingCount;//36 37
ushort BS4_Beams;//38 39

```
ushort BS4_Bins;//40 41
ushort BS4_FirstBinDepthX1000;//42 43
ushort BS4_BinSizeX1000;//44 45
ushort BS4_VolBeginX1000;// 46 47
ushort BS4_VolEndX1000;//48 49
ushort BS4_VolAmpX100;//50 51
ushort BS4_VolThresholdX100;//52 53
ushort BS4_VolGoodPings0;//54 55

//If Backscatter is enabled CWPON[4] 2<CR>
// BACKSCATTER_PROFILE_04
ushort BS4_Amplitude_ID; // 0 1
ushort BS4_Amplitude_Bytes;// 2,3
ushort BS4_AmplitudeX256[bins];// 4  + 2 * bins

// SYSTEM_CHECKSUM
ushort System_Checksum_ID; // 0 1
ushort System_Checksum;//2 3
```

7.6 Ensemble Decode Example C++

```
public class EnsembleClass
 {
    public int MostBins;

    //Header Data
    public ulong Header_Type;
    public ushort Header_PayloadSize;

    //Stage Data
    public ushort Stage_ID;
    public ushort Stage_Bytes;
    public uint Stage_PingCount;
    public float Stage_Range;
    public float Stage_RangeSTD;
    public float Stage_RangeSNR;
    public uint Stage_RangePingsGood;
    public float Stage_Depth;
    public float Stage_DepthSTD;
    public uint Stage_DepthPingsGood;
    public uint Stage_Status;

    //Leader Data
    public ushort System_ID;
    public ushort System_Bytes;
    public uint System_EnsembleNumber;
    public byte[] System_SN = new byte[32];

    public ushort System_FW_MAJOR;
    public ushort System_FW_MINOR;
    public ushort System_FW_REVISION;
```

```
public ushort System_Year;
public byte System_Month;
public byte System_Day;
public byte System_Hour;
public byte System_Minute;
public byte System_Second;
public byte System_Hsec;
public double System_Latitude;
public double System_Longitude;
public float System_DeployDepth;
public float System_DeployHeight;
public ushort System_RightBank;
public float System_Heading;
public float System_Pitch;
public float System_Roll;
public float System_Salinity;
public float System_Temperature;
public float System_BackPlaneTemperature;
        public float System_HeatSink1Temperature;
public float System_Pressure;
public float System_SpeedOfSound;
public ushort System_Status;
public ushort System_Status2;

//Janus Leader Data
public ushort Janus_ID;
public ushort Janus_Bytes;

public float Janus_Frequency;

public float Janus_Diameter;
public float Janus_BeamAngle;

public float Janus_Rcvr1Temperature;
public float Janus_Rcvr2Temperature;
public float Janus_TransmitVolts;

public float Janus_Gain;
public float Janus_TransmitBandwidth;
public float Janus_ReceiveBandwidth;

public float Janus_SampleFrequency;

public ushort Janus_LagSamples;
public ushort Janus_CyclePerElement;
public ushort Janus_NumberOfElements;
public ushort Janus_NumberOfRepeats;

public ushort Janus_Pings;
public ushort Janus_Beams;
```

```
public ushort Janus_Bins;
public float Janus_FirstBin;
public float Janus_BinSize;

public float Janus_CorrelationThreshold;

public float[] Janus_VolBegin = new float[2];
public float[] Janus_VolEnd = new float[2];
public float[] Janus_VolAmp = new float[2];
public float[] Janus_VolVel = new float[2];
public float[] Janus_VolInst = new float[2];
public float Janus_VolThres;
public ushort[] Janus_Voln = new ushort[2];

//Janus Good Velocity Data
public ushort Janus_Good_ID;
public ushort Janus_Good_Bytes;
public ushort[,] Janus_Good_Pings = new ushort[MaxBeams, MaxBins];

//Janus Velocity Data
public ushort Janus_BeamVelocity_ID;
public ushort Janus_BeamVelocity_Bytes;
public float[,] Janus_BeamVelocity = new float[MaxBeams, MaxBins];

//Janus Instrument Velocity Data
public ushort Janus_InstrumentVelocity_ID;
public ushort Janus_InstrumentVelocity_Bytes;
public float[,] Janus_InstrumentVelocity = new float[MaxBeams, MaxBins];

//Janus Amplitude Data
public ushort Janus_Amplitude_ID;
public ushort Janus_Amplitude_Bytes;
public float[,] Janus_Amplitude = new float[MaxBeams, MaxBins];
public float[] Janus_NoiseAmplitude = new float[MaxBeams];

//Janus Correlation Data
public ushort Janus_Correlation_ID;
public ushort Janus_Correlation_Bytes;
public float[,] Janus_Correlation = new float[MaxBeams, MaxBins];

//Backscatter_0 Leader Data
public ushort BS0_ID;
public ushort BS0_Bytes;

public float BS0_Frequency;

public float BS0_Diameter;
public float BS0_BeamAngle;

public float BS0_Rcvr1Temperature;
public float BS0_Rcvr2Temperature;
```

```
public float BS0_TransmitVolts;

public float BS0_Gain;
public float BS0_TransmitBandwidth;
public float BS0_ReceiveBandwidth;

public float BS0_SampleFrequency;

public ushort BS0_LagSamples;
public ushort BS0_CyclePerElement;
public ushort BS0_NumberOfElements;
public ushort BS0_NumberOfRepeats;

public ushort BS0_Pings;
public ushort BS0_Beams;
public ushort BS0_Bins;
public float BS0_FirstBin;
public float BS0_BinSize;

public float[] BS0_VolBegin = new float [2];
public float[] BS0_VolEnd = new float[2];
public float[] BS0_VolAmp = new float[2];
public float BS0_WPVOLthreshold;
public ushort[] BS0_Voln = new ushort[2];

//Backscatter_0 Amplitude Data
public ushort BS0_Amplitude_ID;
public ushort BS0_Amplitude_Bytes;
public float[,] BS0_Amplitude = new float[MaxBeams, MaxBins];
public float[] BS0_NoiseAmplitude = new float[MaxBeams];


//Backscatter_1 Leader Data
public ushort BS1_ID;
public ushort BS1_Bytes;

public float BS1_Frequency;

public float BS1_Diameter;
public float BS1_BeamAngle;
//public float BS1_TransmitPower;
//public float BS1_Absorption;
public float BS1_Rcvr1Temperature;
public float BS1_Rcvr2Temperature;
public float BS1_TransmitVolts;

public float BS1_Gain;
public float BS1_TransmitBandwidth;
public float BS1_ReceiveBandwidth;

public float BS1_SampleFrequency;
```

```
public ushort BS1_LagSamples;
public ushort BS1_CyclePerElement;
public ushort BS1_NumberOfElements;
public ushort BS1_NumberOfRepeats;

public ushort BS1_Pings;
public ushort BS1_Beams;
public ushort BS1_Bins;
public float BS1_FirstBin;
public float BS1_BinSize;

public float[] BS1_VolBegin = new float[2];
public float[] BS1_VolEnd = new float[2];
public float[] BS1_VolAmp = new float[2];
public float BS1_WPVOLthreshold;
public ushort[] BS1_Voln = new ushort[2];

//Backscatter_1 Amplitude Data
public ushort BS1_Amplitude_ID;
public ushort BS1_Amplitude_Bytes;
public float[,] BS1_Amplitude = new float[MaxBeams, MaxBins];
public float[] BS1_NoiseAmplitude = new float[MaxBeams];

//Backscatter_2 Leader Data
public ushort BS2_ID;
public ushort BS2_Bytes;

public float BS2_Frequency;

public float BS2_Diameter;
public float BS2_BeamAngle;
//public float BS2_TransmitPower;
//public float BS2_Absorption;
public float BS2_Rcvr1Temperature;
public float BS2_Rcvr2Temperature;
public float BS2_TransmitVolts;

public float BS2_Gain;
public float BS2_TransmitBandwidth;
public float BS2_ReceiveBandwidth;

public float BS2_SampleFrequency;

public ushort BS2_LagSamples;
public ushort BS2_CyclePerElement;
public ushort BS2_NumberOfElements;
public ushort BS2_NumberOfRepeats;

public ushort BS2_Pings;
public ushort BS2_Beams;
```

```
public ushort BS2_Bins;
public float BS2_FirstBin;
public float BS2_BinSize;

public float[] BS2_VolBegin = new float[2];
public float[] BS2_VolEnd = new float[2];
public float[] BS2_VolAmp = new float[2];
public float BS2_WPVOLthreshold;
public ushort[] BS2_Voln = new ushort[2];

//Backscatter_2 Amplitude Data
public ushort BS2_Amplitude_ID;
public ushort BS2_Amplitude_Bytes;
public float[,] BS2_Amplitude = new float[MaxBeams, MaxBins];
public float[] BS2_NoiseAmplitude = new float[MaxBeams];

//Backscatter_3 Leader Data
public ushort BS3_ID;
public ushort BS3_Bytes;

public float BS3_Frequency;

public float BS3_Diameter;
public float BS3_BeamAngle;
//public float BS3_TransmitPower;
//public float BS3_Absorption;
public float BS3_Rcvr1Temperature;
public float BS3_Rcvr2Temperature;
public float BS3_TransmitVolts;

public float BS3_Gain;
public float BS3_TransmitBandwidth;
public float BS3_ReceiveBandwidth;

public float BS3_SampleFrequency;

public ushort BS3_LagSamples;
public ushort BS3_CyclePerElement;
public ushort BS3_NumberOfElements;
public ushort BS3_NumberOfRepeats;

public ushort BS3_Pings;
public ushort BS3_Beams;
public ushort BS3_Bins;
public float BS3_FirstBin;
public float BS3_BinSize;

public float[] BS3_VolBegin = new float[2];
public float[] BS3_VolEnd = new float[2];
public float[] BS3_VolAmp = new float[2];
public float BS3_WPVOLthreshold;
```

```
public ushort[] BS3_Voln = new ushort[2];

//Backscatter_2 Amplitude Data
public ushort BS3_Amplitude_ID;
public ushort BS3_Amplitude_Bytes;
public float[,] BS3_Amplitude = new float[MaxBeams, MaxBins];
public float[] BS3_NoiseAmplitude = new float[MaxBeams];

//Backscatter_4 Leader Data
public ushort BS4_ID;
public ushort BS4_Bytes;

public float BS4_Frequency;

public float BS4_Diameter;
public float BS4_BeamAngle;
//public float BS4_TransmitPower;
//public float BS4_Absorption;
public float BS4_Rcvr1Temperature;
public float BS4_Rcvr2Temperature;
public float BS4_TransmitVolts;

public float BS4_Gain;
public float BS4_TransmitBandwidth;
public float BS4_ReceiveBandwidth;

public float BS4_SampleFrequency;

public ushort BS4_LagSamples;
public ushort BS4_CyclePerElement;
public ushort BS4_NumberOfElements;
public ushort BS4_NumberOfRepeats;

public ushort BS4_Pings;
public ushort BS4_Beams;
public ushort BS4_Bins;
public float BS4_FirstBin;
public float BS4_BinSize;

public float[] BS4_VolBegin = new float[2];
public float[] BS4_VolEnd = new float[2];
public float[] BS4_VolAmp = new float[2];
public float BS4_WPVOLthreshold;
public ushort[] BS4_Voln = new ushort[2];

//Backscatter_2 Amplitude Data
public ushort BS4_Amplitude_ID;
public ushort BS4_Amplitude_Bytes;
public float[,] BS4_Amplitude = new float[MaxBeams, MaxBins];
public float[] BS4_NoiseAmplitude = new float[MaxBeams];
```

```
    public bool SystemAvailable;
    public bool JanusAvailable;
    public bool JanusBeamGoodAvailable;
    public bool JanusBeamVelocityAvailable;
    public bool JanusInstrumentVelocityAvailable;
    public bool JanusAmplitudeAvailable;
    public bool JanusCorrelationAvailable;

    public bool StageAvailable;

    public bool BS0Available;
    public bool BS0AmplitudeAvailable;
    public bool BS1Available;
    public bool BS1AmplitudeAvailable;
    public bool BS2Available;
    public bool BS2AmplitudeAvailable;
    public bool BS3Available;
    public bool BS3AmplitudeAvailable;
    public bool BS4Available;
    public bool BS4AmplitudeAvailable;
}
```

```
    public struct TestUnion
    {
        [FieldOffset(0)]
        public byte A;
        [FieldOffset(1)]
        public byte B;
        [FieldOffset(2)]
        public byte C;
        [FieldOffset(3)]
        public byte D;
        [FieldOffset(4)]
        public byte E;
        [FieldOffset(5)]
        public byte F;
        [FieldOffset(6)]
        public byte G;
        [FieldOffset(7)]
        public byte H;
        [FieldOffset(0)]
        public short Short;
        [FieldOffset(0)]
        public float Float;
        [FieldOffset(0)]
        public int Int;
        [FieldOffset(0)]
        public long Long;
        [FieldOffset(0)]
        public double Double;
    }

    public static TestUnion ByteArrayToNumber;
```

```
public static int ByteArrayToInt(byte[] packet)
{
    ByteArrayToNumber.A = packet[PacketPointer++];
    ByteArrayToNumber.B = packet[PacketPointer++];
    ByteArrayToNumber.C = packet[PacketPointer++];
    ByteArrayToNumber.D = packet[PacketPointer++];

    return ByteArrayToNumber.Int;
}
public static short ByteArrayToShort(byte[] packet)
{
    ByteArrayToNumber.A = packet[PacketPointer++];
    ByteArrayToNumber.B = packet[PacketPointer++];

    return ByteArrayToNumber.Short;
}
public static long ByteArrayToLong(byte[] packet)
{
    ByteArrayToNumber.A = packet[PacketPointer++];
    ByteArrayToNumber.B = packet[PacketPointer++];
    ByteArrayToNumber.C = packet[PacketPointer++];
    ByteArrayToNumber.D = packet[PacketPointer++];
    ByteArrayToNumber.E = packet[PacketPointer++];
    ByteArrayToNumber.F = packet[PacketPointer++];
    ByteArrayToNumber.G = packet[PacketPointer++];
    ByteArrayToNumber.H = packet[PacketPointer++];

    return ByteArrayToNumber.Long;
}
public static string ByteArrayToString(byte[] packet, int len)
{
    string s = "";
    int i;
    for (i = 0; i < len; i++)
    {
        s += (char)packet[PacketPointer++];
    }
    return s;
}
public static float ByteArrayToFloat(byte[] packet)
{
    ByteArrayToNumber.A = packet[PacketPointer++];
    ByteArrayToNumber.B = packet[PacketPointer++];
    ByteArrayToNumber.C = packet[PacketPointer++];
    ByteArrayToNumber.D = packet[PacketPointer++];

    return ByteArrayToNumber.Float;
}
public static double ByteArrayToDouble(byte[] packet)
{
    ByteArrayToNumber.A = packet[PacketPointer++];
    ByteArrayToNumber.B = packet[PacketPointer++];
    ByteArrayToNumber.C = packet[PacketPointer++];
    ByteArrayToNumber.D = packet[PacketPointer++];

    ByteArrayToNumber.E = packet[PacketPointer++];
    ByteArrayToNumber.F = packet[PacketPointer++];
    ByteArrayToNumber.G = packet[PacketPointer++];
```

```
        ByteArrayToNumber.H = packet[PacketPointer++];

        return ByteArrayToNumber.Double;
    }

public static void DecodeEnsemble(byte[] packet, EnsembleClass Ensemble)
{
   int i, beam, bin;

   for(i=0;i< MaxDataTypes; i++)
      DataTypeAvailable[i] = false;

   //init just in case the ID are out of order
   Ensemble.Janus_Beams = 0;// MaxBeams;
   Ensemble.Janus_Bins = 0;// MaxBins;

   Ensemble.BS0_Beams = 0;// MaxBeams;
   Ensemble.BS0_Bins = 0;// MaxBins;

   Ensemble.BS1_Beams = 0;// MaxBeams;
   Ensemble.BS1_Bins = 0;// MaxBins;

   Ensemble.BS2_Beams = 0;// MaxBeams;
   Ensemble.BS2_Bins = 0;// MaxBins;

   Ensemble.BS3_Beams = 0;// MaxBeams;
   Ensemble.BS3_Bins = 0;// MaxBins;

   Ensemble.BS4_Beams = 0;// MaxBeams;
   Ensemble.BS4_Bins = 0;// MaxBins;

   Ensemble.SystemAvailable = false;
   Ensemble.JanusAvailable = false;
   Ensemble.JanusBeamGoodAvailable = false;
   Ensemble.JanusBeamVelocityAvailable = false;
   Ensemble.JanusInstrumentVelocityAvailable = false;
   Ensemble.JanusAmplitudeAvailable = false;
   Ensemble.JanusCorrelationAvailable = false;

   Ensemble.StageAvailable = false;

   Ensemble.BS0Available = false;
   Ensemble.BS0AmplitudeAvailable = false;

   Ensemble.BS1Available = false;
   Ensemble.BS1AmplitudeAvailable = false;

   Ensemble.BS2Available = false;
   Ensemble.BS2AmplitudeAvailable = false;

   Ensemble.BS3Available = false;
```

```
Ensemble.BS3AmplitudeAvailable = false;

Ensemble.BS4Available = false;
Ensemble.BS4AmplitudeAvailable = false;

Ensemble.MostBins = 0;

PacketPointer = 0;//point to first byte of the data buffer

//get Header
Ensemble.Header_Type = (ulong)ByteArrayToLong(packet);
Ensemble.Header_PayloadSize = (ushort)ByteArrayToShort(packet);
PacketPointer += 2;

//decode the linked list
uint PayloadStart = PacketPointer;
uint NextID = PacketPointer;
bool done = false;
ushort TempBytes;
while (!done)
{
   PacketPointer = NextID;
   ushort ID = (ushort)ByteArrayToShort(packet);

   if(ID < MaxDataTypes)
      DataTypeAvailable[ID] = true;

   switch (ID)
   {
      case SYSTEM_CHECKSUM:
         done = true;
         break;
      default://unknown ID
         TempBytes = (ushort)ByteArrayToShort(packet);
         NextID = PacketPointer + TempBytes;
         break;
      case VERTICAL_STAGE:
         Ensemble.StageAvailable = true;

         Ensemble.Stage_ID = ID;
         Ensemble.Stage_Bytes = (ushort)ByteArrayToShort(packet);
         NextID = PacketPointer + Ensemble.Stage_Bytes;

         Ensemble.Stage_PingCount = (ushort)ByteArrayToShort(packet);

         Ensemble.Stage_Range = ByteArrayToFloat(packet);
         Ensemble.Stage_RangeSTD = ByteArrayToFloat(packet);
         Ensemble.Stage_RangeSNR = ByteArrayToFloat(packet);

         Ensemble.Stage_RangePingsGood = (ushort)ByteArrayToShort(packet);
```

```
    Ensemble.Stage_Depth = ByteArrayToFloat(packet);
    Ensemble.Stage_DepthSTD = ByteArrayToFloat(packet);

    Ensemble.Stage_DepthPingsGood = (ushort)ByteArrayToShort(packet);
    Ensemble.Stage_Status = (ushort)ByteArrayToShort(packet);

    break;
  case SYSTEM_LEADER:
    Ensemble.SystemAvailable = true;

    Ensemble.System_ID = ID;
    Ensemble.System_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.System_Bytes;

    Ensemble.System_EnsembleNumber = (uint)ByteArrayToInt(packet);

    for (i = 0; i < 32; i++)
    {
       Ensemble.System_SN[i] = (byte)packet[PacketPointer++];
    }

    Ensemble.System_FW_MAJOR = (ushort)ByteArrayToShort(packet);
    Ensemble.System_FW_MINOR = (ushort)ByteArrayToShort(packet);
    Ensemble.System_FW_REVISION = (ushort)ByteArrayToShort(packet);

    Ensemble.System_Year = (ushort)ByteArrayToShort(packet);
    Ensemble.System_Month = (byte)packet[PacketPointer++];
    Ensemble.System_Day = (byte)packet[PacketPointer++];
    Ensemble.System_Hour = (byte)packet[PacketPointer++];
    Ensemble.System_Minute = (byte)packet[PacketPointer++];
    Ensemble.System_Second = (byte)packet[PacketPointer++];
    Ensemble.System_Hsec = (byte)packet[PacketPointer++];

    Ensemble.System_Latitude = ByteArrayToDouble(packet);
    Ensemble.System_Longitude = ByteArrayToDouble(packet);
    Ensemble.System_DeployDepth = ByteArrayToFloat(packet);
    Ensemble.System_DeployHeight = ByteArrayToFloat(packet);

    Ensemble.System_RightBank = (ushort)ByteArrayToShort(packet);

    Ensemble.System_Heading = (float)((ushort)ByteArrayToShort(packet)) / 100;
    Ensemble.System_Pitch = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_Roll = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_Salinity = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_BackPlaneTemperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_HeatSink1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.System_Pressure = ByteArrayToFloat(packet);
    Ensemble.System_SpeedOfSound = ByteArrayToFloat(packet);
    Ensemble.System_Status = (ushort)ByteArrayToShort(packet);
    Ensemble.System_Status2 = (ushort)ByteArrayToShort(packet);
```

```
    PacketPointer = NextID;
    break;
  case HORIZONTAL_JANUS_LEADER:
    Ensemble.JanusAvailable = true;

    Ensemble.Janus_ID = ID;
    Ensemble.Janus_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.Janus_Bytes;

    Ensemble.Janus_Frequency = ByteArrayToFloat(packet);
    Ensemble.Janus_Diameter = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.Janus_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.Janus_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.Janus_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.Janus_Gain = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.Janus_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.Janus_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.Janus_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.Janus_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.Janus_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.Janus_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.Janus_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.Janus_Beams = (ushort)ByteArrayToShort(packet);
    if (Ensemble.Janus_Beams > MaxBeams)
      Ensemble.Janus_Beams = MaxBeams;
    Ensemble.Janus_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.Janus_Bins > MaxBins)
      Ensemble.Janus_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.Janus_Bins)
      Ensemble.MostBins = Ensemble.Janus_Bins;

    Ensemble.Janus_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_BinSize = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.Janus_CorrelationThreshold = (float)ByteArrayToShort(packet) / 1000;


    Ensemble.Janus_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.Janus_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.Janus_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
```

```
            Ensemble.Janus_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;

            Ensemble.Janus_VolVel[0] = (float)ByteArrayToShort(packet) / 1000;
            Ensemble.Janus_VolVel[1] = (float)ByteArrayToShort(packet) / 1000;

            Ensemble.Janus_VolInst[0] = (float)ByteArrayToShort(packet) / 2000;
            Ensemble.Janus_VolInst[1] = (float)ByteArrayToShort(packet) / 2000;

            Ensemble.Janus_VolThres = (float)ByteArrayToShort(packet) / 100;

            Ensemble.Janus_Voln[0] = (ushort)ByteArrayToShort(packet);
            Ensemble.Janus_Voln[1] = (ushort)ByteArrayToShort(packet);

            PacketPointer = NextID;
            break;

        case HORIZONTAL_JANUS_GOOD:
            Ensemble.JanusBeamGoodAvailable = true;

            Ensemble.Janus_Good_ID = ID;
            Ensemble.Janus_Good_Bytes = (ushort)ByteArrayToShort(packet);
            NextID = PacketPointer + Ensemble.Janus_Good_Bytes;

            for (bin = 0; bin < Ensemble.Janus_Bins; bin++)
            {
                for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
                {
                    Ensemble.Janus_Good_Pings[beam, bin] = (ushort)ByteArrayToShort(packet);
                }
            }
            break;
        case HORIZONTAL_JANUS_BEAM:
            Ensemble.JanusBeamVelocityAvailable = true;

            Ensemble.Janus_BeamVelocity_ID = ID;
            Ensemble.Janus_BeamVelocity_Bytes = (ushort)ByteArrayToShort(packet);
            NextID = PacketPointer + Ensemble.Janus_BeamVelocity_Bytes;

            for (bin = 0; bin < Ensemble.Janus_Bins; bin++)
            {
                for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
                {
                    Ensemble.Janus_BeamVelocity[beam, bin] = (float)ByteArrayToShort(packet) /
2000;
                }
            }
            break;
        case HORIZONTAL_JANUS_INSTRUMENT:
            Ensemble.JanusInstrumentVelocityAvailable = true;

            Ensemble.Janus_InstrumentVelocity_ID = ID;
```

```
            Ensemble.Janus_InstrumentVelocity_Bytes = (ushort)ByteArrayToShort(packet);
            NextID = PacketPointer + Ensemble.Janus_InstrumentVelocity_Bytes;

            for (bin = 0; bin < Ensemble.Janus_Bins; bin++)
            {
               for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
               {
                  Ensemble.Janus_InstrumentVelocity[beam, bin] = (float)ByteArrayToShort(packet) /
2000;
               }
            }
            break;
         case HORIZONTAL_JANUS_AMPLITUDE:
            Ensemble.JanusAmplitudeAvailable = true;

            Ensemble.Janus_Amplitude_ID = ID;
            Ensemble.Janus_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
            NextID = PacketPointer + Ensemble.Janus_Amplitude_Bytes;

            for (bin = 0; bin < Ensemble.Janus_Bins; bin++)
            {
               for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
               {
                  Ensemble.Janus_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
               }
            }
            for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
            {
               Ensemble.Janus_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
            }

            break;
         case HORIZONTAL_JANUS_CORRELATION:
            Ensemble.JanusCorrelationAvailable = true;

            Ensemble.Janus_Correlation_ID = ID;
            Ensemble.Janus_Correlation_Bytes = (ushort)ByteArrayToShort(packet);
            NextID = PacketPointer + Ensemble.Janus_Correlation_Bytes;

            for (bin = 0; bin < Ensemble.Janus_Bins; bin++)
            {
               for (beam = 0; beam < Ensemble.Janus_Beams; beam++)
               {
                  Ensemble.Janus_Correlation[beam, bin] = (float)ByteArrayToShort(packet) / 16384;
               }
            }
            break;
         case BACKSCATTER_LEADER_00:
            Ensemble.BS0Available = true;

            Ensemble.BS0_ID = ID;
```

```
    Ensemble.BS0_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS0_Bytes;

    Ensemble.BS0_Frequency = ByteArrayToFloat(packet);
    Ensemble.BS0_Diameter = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS0_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS0_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS0_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS0_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.BS0_Gain = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS0_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.BS0_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.BS0_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.BS0_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.BS0_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.BS0_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.BS0_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.BS0_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.BS0_Beams = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS0_Beams > MaxBeams)
       Ensemble.BS0_Beams = MaxBeams;
    Ensemble.BS0_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS0_Bins > MaxBins)
       Ensemble.BS0_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.BS0_Bins)
       Ensemble.MostBins = Ensemble.BS0_Bins;
    Ensemble.BS0_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS0_BinSize = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS0_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    if(Ensemble.BS0_Beams > 1)
       Ensemble.BS0_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS0_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS0_Beams > 1)
       Ensemble.BS0_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS0_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
    if (Ensemble.BS0_Beams > 1)
       Ensemble.BS0_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS0_WPVOLthreshold = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS0_Voln[0] = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS0_Beams > 1)
       Ensemble.BS0_Voln[1] = (ushort)ByteArrayToShort(packet);

    PacketPointer = NextID;
    break;
 case BACKSCATTER_PROFILE_00:
```

```
    Ensemble.BS0AmplitudeAvailable = true;

    Ensemble.BS0_Amplitude_ID = ID;
    Ensemble.BS0_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS0_Amplitude_Bytes;

    for (bin = 0; bin < Ensemble.BS0_Bins; bin++)
    {
       for (beam = 0; beam < Ensemble.BS0_Beams; beam++)
       {
          Ensemble.BS0_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
       }
    }
    for (beam = 0; beam < Ensemble.BS0_Beams; beam++)
    {
       Ensemble.BS0_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
    }
    break;
 case BACKSCATTER_LEADER_01:
    Ensemble.BS1Available = true;

    Ensemble.BS1_ID = ID;
    Ensemble.BS1_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS1_Bytes;

    Ensemble.BS1_Frequency = ByteArrayToFloat(packet);
    Ensemble.BS1_Diameter = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS1_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS1_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS1_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS1_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.BS1_Gain = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS1_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.BS1_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.BS1_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.BS1_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.BS1_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.BS1_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.BS1_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.BS1_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.BS1_Beams = (ushort)ByteArrayToShort(packet);

    if (Ensemble.BS1_Beams > MaxBeams)
       Ensemble.BS1_Beams = MaxBeams;
    Ensemble.BS1_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS1_Bins > MaxBins)
```

```
      Ensemble.BS1_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.BS1_Bins)
      Ensemble.MostBins = Ensemble.BS1_Bins;

    Ensemble.BS1_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS1_BinSize = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS1_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS1_Beams > 1)
      Ensemble.BS1_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS1_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS1_Beams > 1)
      Ensemble.BS1_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS1_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
    if (Ensemble.BS1_Beams > 1)
      Ensemble.BS1_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS1_WPVOLthreshold = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS1_Voln[0] = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS1_Beams > 1)
      Ensemble.BS1_Voln[1] = (ushort)ByteArrayToShort(packet);

    PacketPointer = NextID;
    break;
  case BACKSCATTER_PROFILE_01:
    Ensemble.BS1AmplitudeAvailable = true;

    Ensemble.BS1_Amplitude_ID = ID;
    Ensemble.BS1_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS1_Amplitude_Bytes;

    for (bin = 0; bin < Ensemble.BS1_Bins; bin++)
    {
      for (beam = 0; beam < Ensemble.BS1_Beams; beam++)
      {
        Ensemble.BS1_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
      }
    }
    for (beam = 0; beam < Ensemble.BS1_Beams; beam++)
    {
      Ensemble.BS1_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
    }
    break;
  case BACKSCATTER_LEADER_02:
    Ensemble.BS2Available = true;

    Ensemble.BS2_ID = ID;
    Ensemble.BS2_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS2_Bytes;

    Ensemble.BS2_Frequency = ByteArrayToFloat(packet);
    Ensemble.BS2_Diameter = (float)ByteArrayToShort(packet) / 1000;
```

```
    Ensemble.BS2_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS2_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS2_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS2_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.BS2_Gain = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS2_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.BS2_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.BS2_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.BS2_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.BS2_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.BS2_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.BS2_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.BS2_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.BS2_Beams = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS2_Beams > MaxBeams)
       Ensemble.BS2_Beams = MaxBeams;
    Ensemble.BS2_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS2_Bins > MaxBins)
       Ensemble.BS2_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.BS2_Bins)
       Ensemble.MostBins = Ensemble.BS2_Bins;

    Ensemble.BS2_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS2_BinSize = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS2_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS2_Beams > 1)
       Ensemble.BS2_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS2_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS2_Beams > 1)
       Ensemble.BS2_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS2_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
    if (Ensemble.BS2_Beams > 1)
       Ensemble.BS2_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS2_WPVOLthreshold = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS2_Voln[0] = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS2_Beams > 1)
       Ensemble.BS2_Voln[1] = (ushort)ByteArrayToShort(packet);

    break;
  case BACKSCATTER_PROFILE_02:
    Ensemble.BS2AmplitudeAvailable = true;

    Ensemble.BS2_Amplitude_ID = ID;
    Ensemble.BS2_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS2_Amplitude_Bytes;
```

```
    for (bin = 0; bin < Ensemble.BS2_Bins; bin++)
    {
       for (beam = 0; beam < Ensemble.BS2_Beams; beam++)
       {
          Ensemble.BS2_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
       }
    }
    for (beam = 0; beam < Ensemble.BS2_Beams; beam++)
    {
       Ensemble.BS2_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
    }
    break;
 case BACKSCATTER_LEADER_03:
    Ensemble.BS3Available = true;

    Ensemble.BS3_ID = ID;
    Ensemble.BS3_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS3_Bytes;

    Ensemble.BS3_Frequency = ByteArrayToFloat(packet);
    Ensemble.BS3_Diameter = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS3_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS3_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS3_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS3_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.BS3_Gain = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS3_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.BS3_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.BS3_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.BS3_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.BS3_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.BS3_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.BS3_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.BS3_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.BS3_Beams = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS3_Beams > MaxBeams)
       Ensemble.BS3_Beams = MaxBeams;
    Ensemble.BS3_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS3_Bins > MaxBins)
       Ensemble.BS3_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.BS3_Bins)
       Ensemble.MostBins = Ensemble.BS3_Bins;

    Ensemble.BS3_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS3_BinSize = (float)ByteArrayToShort(packet) / 1000;
```

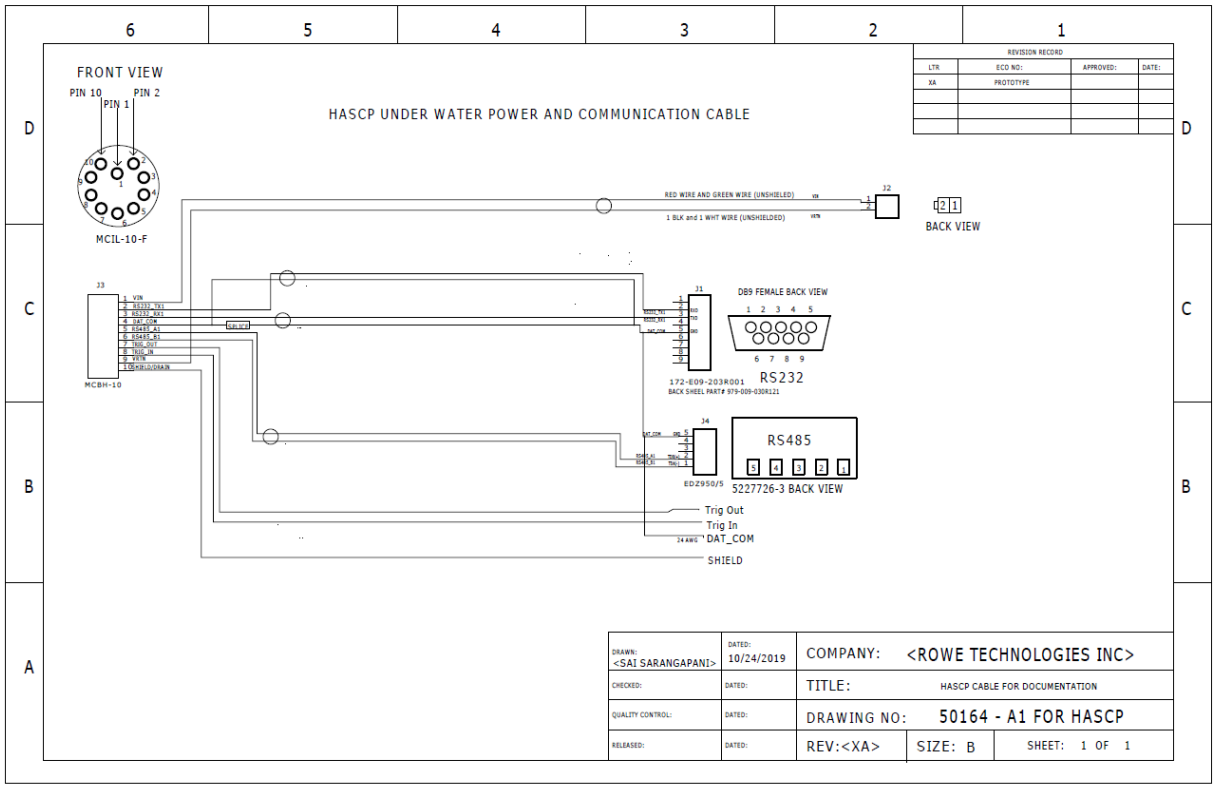
```
    Ensemble.BS3_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS3_Beams > 1)
       Ensemble.BS3_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS3_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS3_Beams > 1)
       Ensemble.BS3_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS3_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
    if (Ensemble.BS3_Beams > 1)
       Ensemble.BS3_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS3_WPVOLthreshold = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS3_Voln[0] = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS3_Beams > 1)
       Ensemble.BS3_Voln[1] = (ushort)ByteArrayToShort(packet);

    break;
 case BACKSCATTER_PROFILE_03:
    Ensemble.BS3AmplitudeAvailable = true;

    Ensemble.BS3_Amplitude_ID = ID;
    Ensemble.BS3_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS3_Amplitude_Bytes;

    for (bin = 0; bin < Ensemble.BS3_Bins; bin++)
    {
       for (beam = 0; beam < Ensemble.BS3_Beams; beam++)
       {
          Ensemble.BS3_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
       }
    }
    for (beam = 0; beam < Ensemble.BS3_Beams; beam++)
    {
       Ensemble.BS3_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
    }
    break;
 case BACKSCATTER_LEADER_04:
    Ensemble.BS4Available = true;

    Ensemble.BS4_ID = ID;
    Ensemble.BS4_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS2_Bytes;

    Ensemble.BS4_Frequency = ByteArrayToFloat(packet);
    Ensemble.BS4_Diameter = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS4_BeamAngle = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS4_Rcvr1Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS4_Rcvr2Temperature = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS4_TransmitVolts = (float)ByteArrayToShort(packet) / 100;

    Ensemble.BS4_Gain = (float)ByteArrayToShort(packet) / 1000;
```

```
    Ensemble.BS4_TransmitBandwidth = (float)ByteArrayToShort(packet) / 65535;
    Ensemble.BS4_ReceiveBandwidth = (float)ByteArrayToShort(packet) / 65535;

    Ensemble.BS4_SampleFrequency = ByteArrayToFloat(packet);

    Ensemble.BS4_LagSamples = (ushort)ByteArrayToShort(packet);
    Ensemble.BS4_CyclePerElement = (ushort)ByteArrayToShort(packet);
    Ensemble.BS4_NumberOfElements = (ushort)ByteArrayToShort(packet);
    Ensemble.BS4_NumberOfRepeats = (ushort)ByteArrayToShort(packet);

    Ensemble.BS4_Pings = (ushort)ByteArrayToShort(packet);
    Ensemble.BS4_Beams = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS4_Beams > MaxBeams)
       Ensemble.BS4_Beams = MaxBeams;
    Ensemble.BS4_Bins = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS4_Bins > MaxBins)
       Ensemble.BS4_Bins = MaxBins;
    if (Ensemble.MostBins < Ensemble.BS4_Bins)
       Ensemble.MostBins = Ensemble.BS4_Bins;

    Ensemble.BS4_FirstBin = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS4_BinSize = (float)ByteArrayToShort(packet) / 1000;

    Ensemble.BS4_VolBegin[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS4_Beams > 1)
       Ensemble.BS4_VolBegin[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS4_VolEnd[0] = (float)ByteArrayToShort(packet) / 1000;
    if (Ensemble.BS4_Beams > 1)
       Ensemble.BS4_VolEnd[1] = (float)ByteArrayToShort(packet) / 1000;
    Ensemble.BS4_VolAmp[0] = (float)ByteArrayToShort(packet) / 100;
    if (Ensemble.BS4_Beams > 1)
       Ensemble.BS4_VolAmp[1] = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS4_WPVOLthreshold = (float)ByteArrayToShort(packet) / 100;
    Ensemble.BS4_Voln[0] = (ushort)ByteArrayToShort(packet);
    if (Ensemble.BS4_Beams > 1)
       Ensemble.BS4_Voln[1] = (ushort)ByteArrayToShort(packet);

    break;
  case BACKSCATTER_PROFILE_04:
    Ensemble.BS4AmplitudeAvailable = true;

    Ensemble.BS4_Amplitude_ID = ID;
    Ensemble.BS4_Amplitude_Bytes = (ushort)ByteArrayToShort(packet);
    NextID = PacketPointer + Ensemble.BS4_Amplitude_Bytes;

    for (bin = 0; bin < Ensemble.BS4_Bins; bin++)
    {
       for (beam = 0; beam < Ensemble.BS4_Beams; beam++)
       {
          Ensemble.BS4_Amplitude[beam, bin] = (float)ByteArrayToShort(packet) / 256;
       }
```

```
            }
            for (beam = 0; beam < Ensemble.BS4_Beams; beam++)
            {
               Ensemble.BS4_NoiseAmplitude[beam] = (float)ByteArrayToShort(packet) / 256;
            }
            break;
      }
      if (NextID > PayloadStart + Ensemble.Header_PayloadSize)
         done = true;
      else
          PacketPointer = NextID;
   }
}
```

8 Cable Wiring Diagrams

This section outlines the HASCP communication cable wiring.



9 Few pics of the HASCP taken during testing at RTI and at lake.

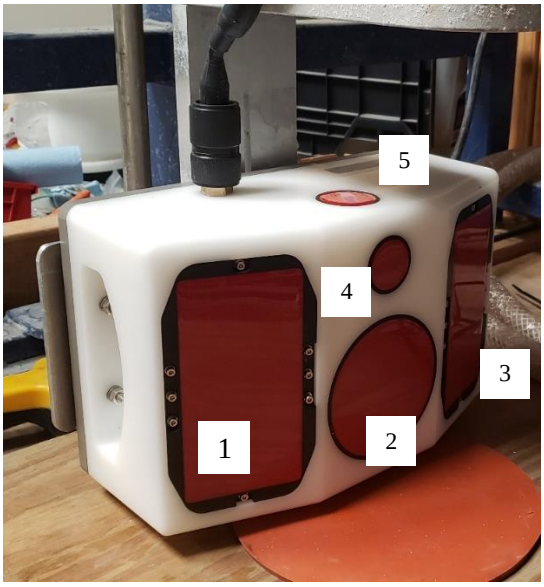


Figure 16 Isometric view of HASCP showing two 1200 kHz rectangular transducers (beams 1,3), 4 inch 600 kHz piston transducer (beam 2), 2.4 MHz piston transducer (beam 4) and a 1200 kHz piston transducer on top (beam 5).

Figure 17. Front view of the HASCP showing the two-1200 kHz rectangular transducers, a single-4 inch 600 kHz piston transducer and a single - 2.4 MHz piston transducer.

Figure 18. Fixture used for mounting the HASCP at the lake. The instrument was mounted at an angle of 45 degrees with all the beams facing down at the lake.

Figure 19. Another view of the fixture used for mounting the HASCP at the lake. The instrument was mounted at an angle of 45 degrees with all the beams facing down at the lake.

Figure 20 Mounting the HASCP in the Rowe boat for lake testing showing the HASCP in water

9.1 Example Plots from Lake Test

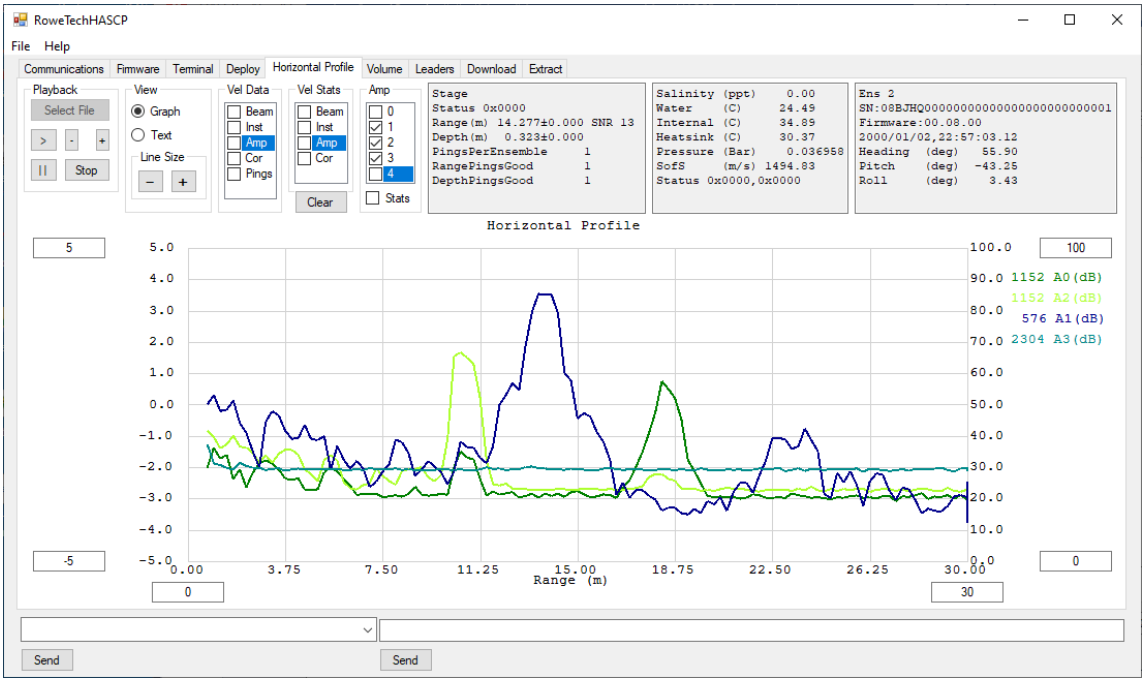


Figure 21. Plot of the horizontal profile showing the amplitude measurements by the various beams plotted across the range in m. This data was measured by the HASCP at the lake. The left hand axis is the velocity range (-5 to + 5 m/s) and the right hand axis represents the RSSI measured by the ADCP from 0 to 100 dB. The numbers on the axis can be changed by typing into the box.

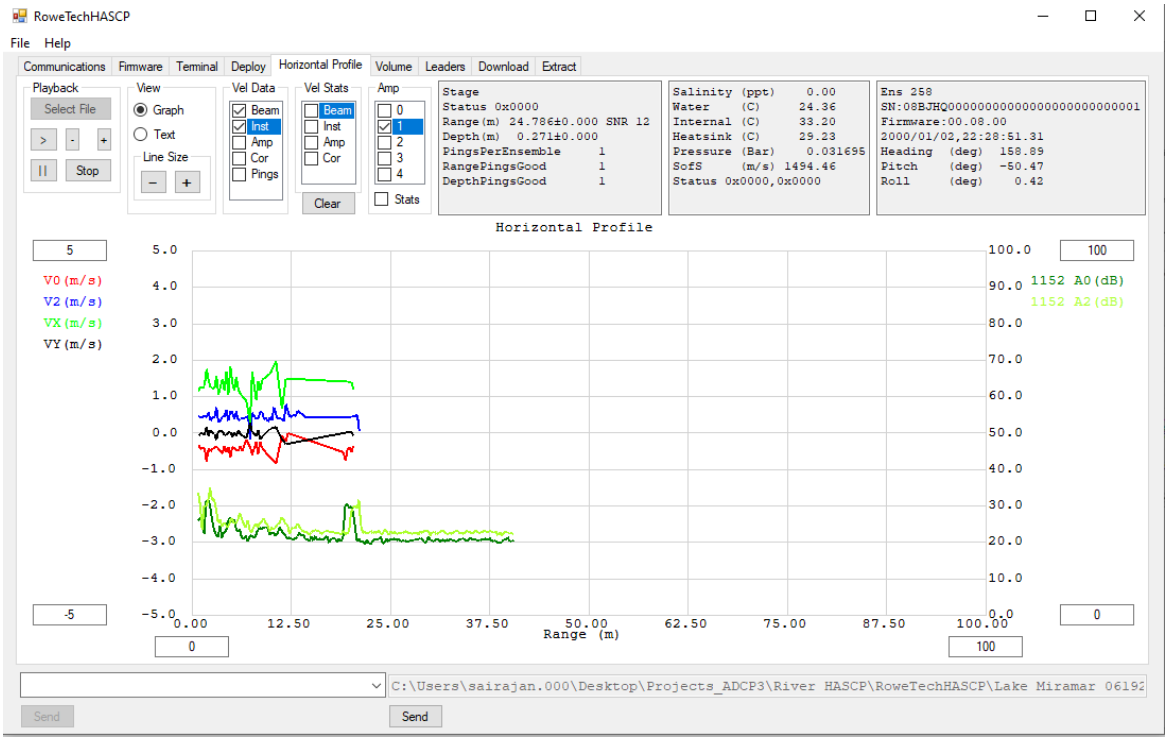


Figure 22. Plot of the RSSI , Velocity (Beam and XYZ) profiles measured by the two 1200 kHz rectangular pistons at the lake test. V0 and V2 are the velocity measured by the Beam 0 and Beam 2 respectively in beam coordinates and Vx and Vy are the velocities in XY coordinates. A0 and A2 are the RSSI measured by the Beam 0 and Beam 2 respectively. Beam 0 was pointed in the reverse direction and Beam 2 was pointed in the positive direction.

10 Mechanical Drawings and Assembly

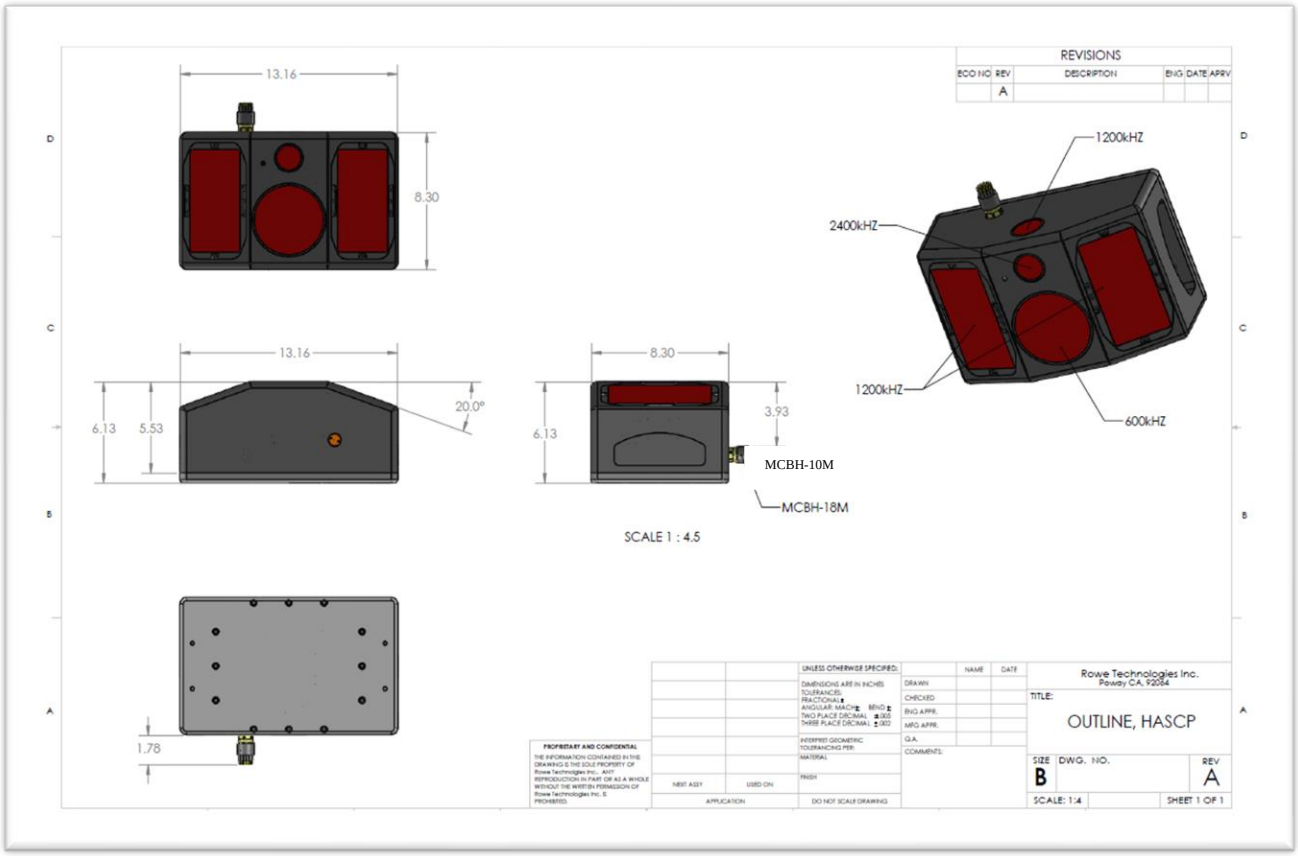


Figure 23. Outline drawing of the HASCP unit.

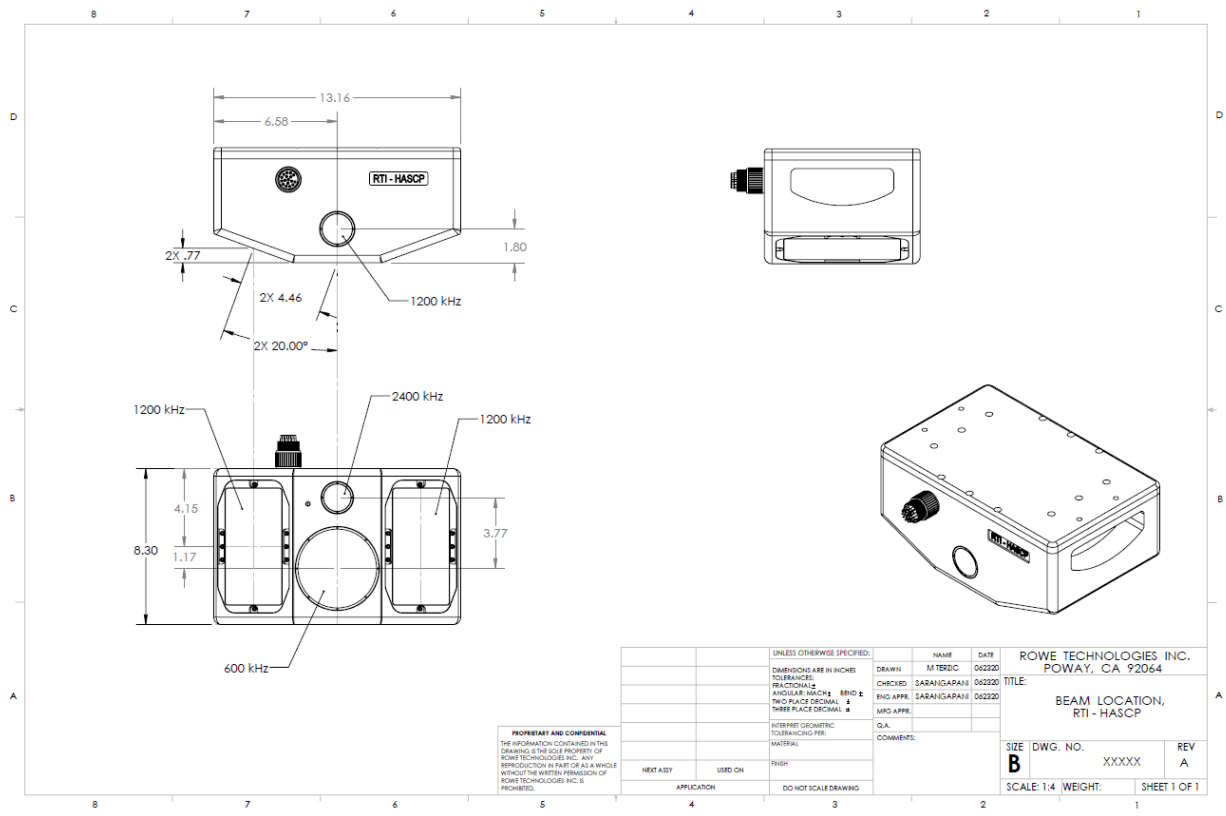


Figure 24. Location of transducer beams in the HASCP unit.

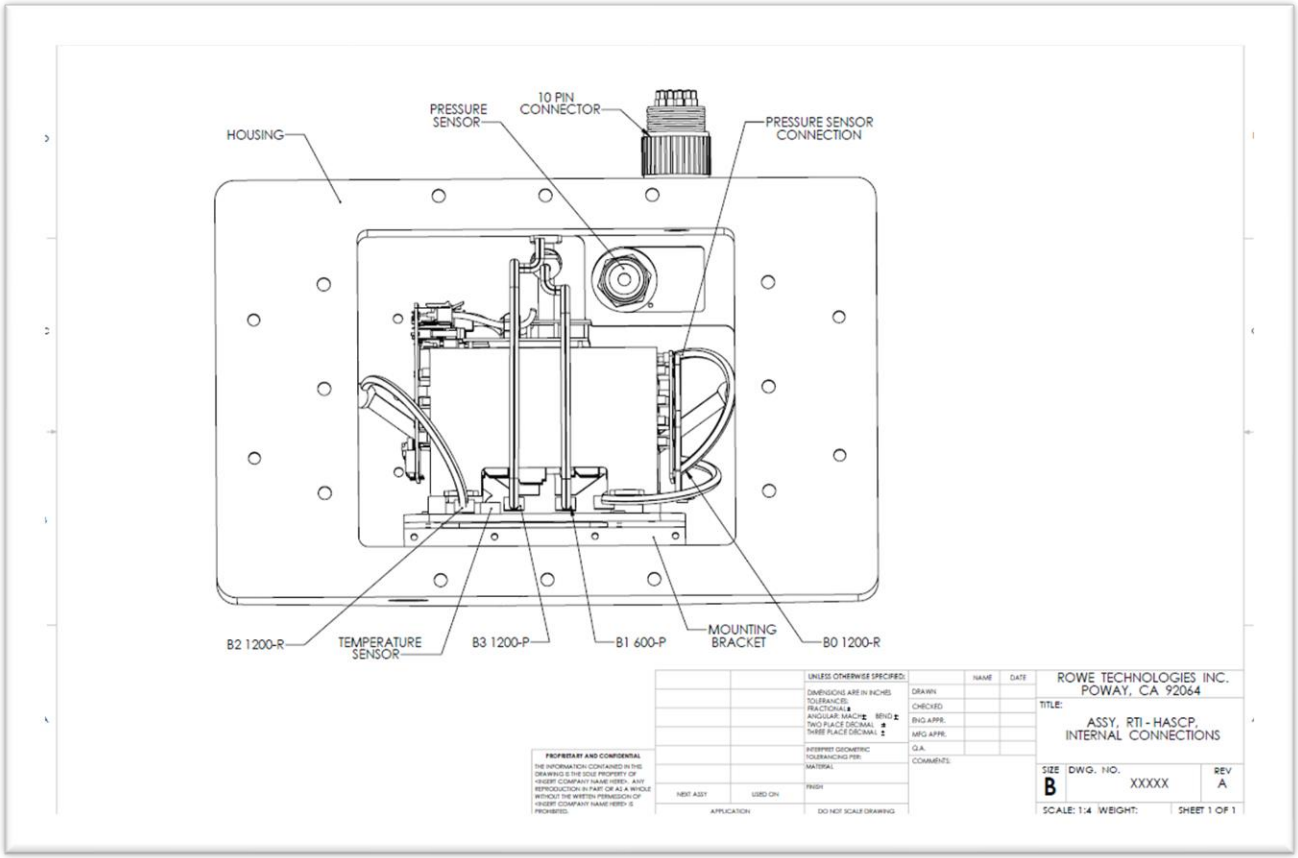


Figure 25. Back View of the HASCP with the back plate open showing the mounting brackets used for mounting the electronics stack in the HASCP. The wiring for the various transducer and sensors inside the unit are also shown.

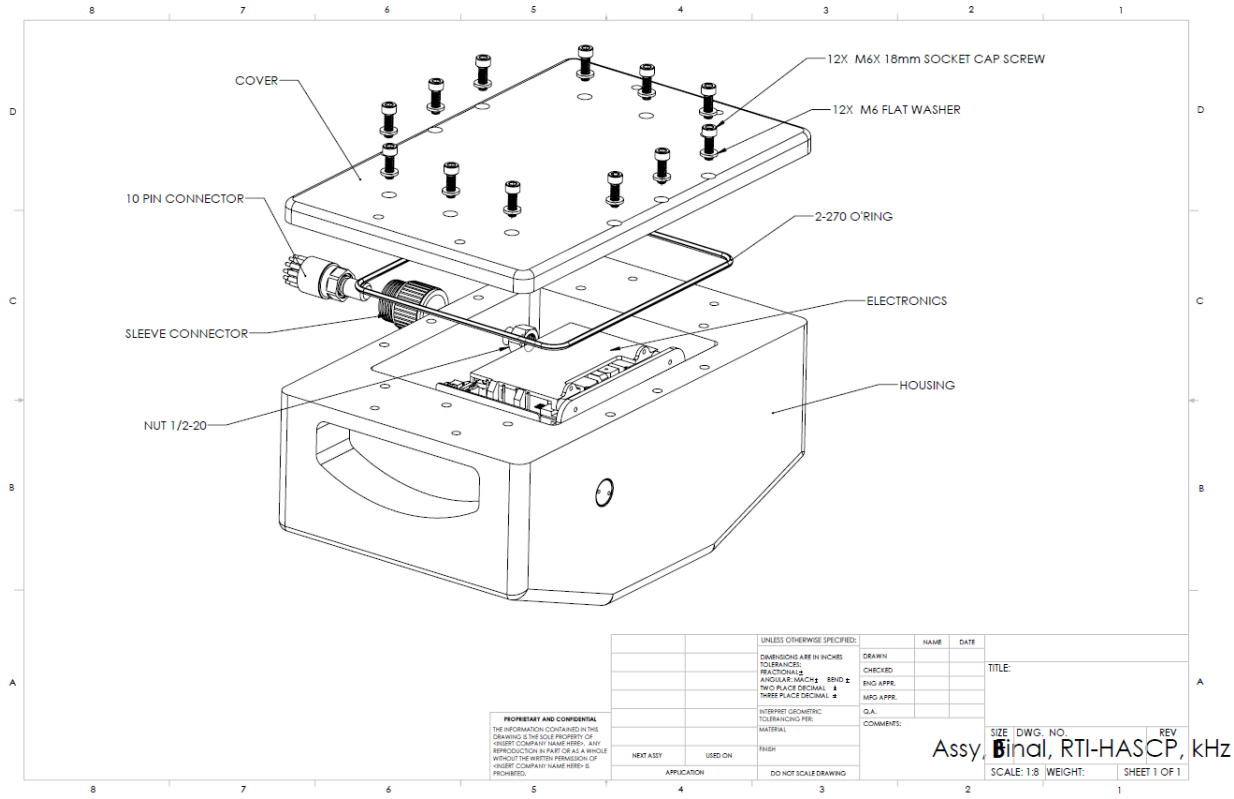


Figure 26. Exploded view of the HASCP unit showing the details of the hardware used.

10.1 Installation to a pole

The HASCP unit was tested in the RTI tank using the 3 inch diameter pole and mounting bracket. The drawing of the mounting bracket used for securing the HASCP on a 3 inch diameter pole is shown below in Figure 27.

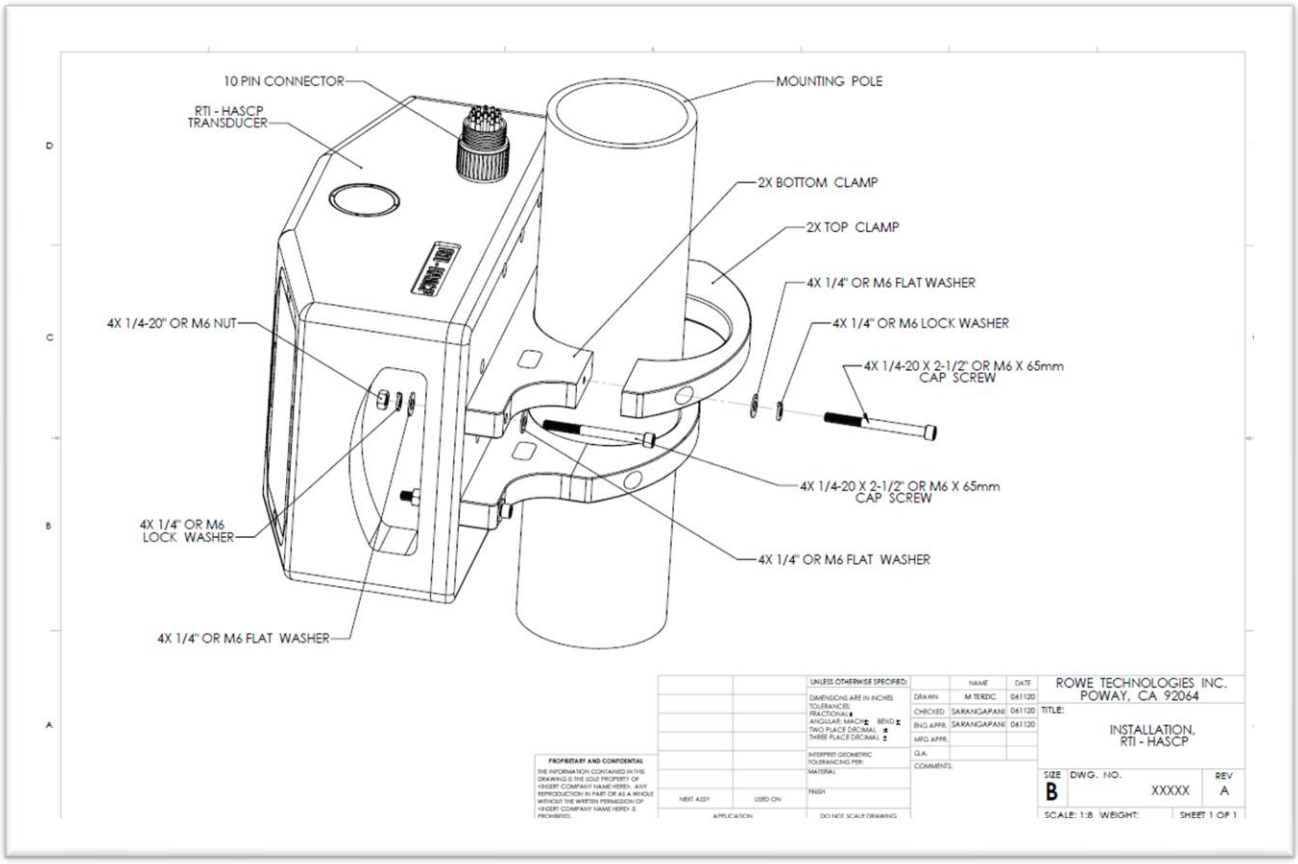


Figure 27. Installation drawing of the RTI-HASCP on the 3 inch diameter pole using the mounting brackets.

Figure 28. HASCP mounted on a pole used for internal testing at RTI.

10.2 Warranty Policy

Equipment manufactured by Rowe Technologies Inc., (ROWETECH) is covered under a 12 month, limited warranty, which begins from the date of original shipment. This warranty extends to all parts and labor for any malfunction caused by defects in material or errors in workmanship that occurred during the manufacturing process. Any third party items incorporated into, or included with the equipment will bear the warranty of their manufacturer.

This warranty is based upon proper operation and maintenance of the equipment, as detailed in the User's Guide, and does not apply to goods that have been subject to shipping damage, improper installation, misuse/neglect, alteration, damaged during use, or the like. The warranty does not cover deficiencies with the design of the equipment or any damages that are a result of measurement errors from the equipment.

Upon notification of the nature of the defect, ROWETECH will ask you to either return the equipment to a service facility for repair, or we will ship you replacement parts. If the equipment needs to be returned, ROWETECH will provide you with a return merchandise authorization (RMA) number. The equipment should be shipped in the original packaging, with all delivery costs, including duties, taxes, etc., covered by the customer. ROWETECH reserves the right to refuse receipt of equipment returned without a valid RMA number.