Appendix IX: R2Sonic Data Format

15 APPENDIX IX: R2Sonic Uplink Data Formats

15.1 Introduction

This describes the data formats sent from the sonar head and SIM. Unless noted, the data packets are sent from the sonar head. The formats are given in pseudo C.

Head firmware versions 13-Dec-2011, and newer, utilise the data formats in this document. Previous head firmware versions back to 25-Mar-2010 only utilise data formats from sections 15.5 and 15.6 in this document. Future versions of firmware will adhere to this format and may include additional information.

The data format, in older versions of sonar head firmware, is different than the format described in this document and is unsupported.

15.2 General Notes

1. Each info or data section includes a name/size mini-header to allow the parser to easily skip unneeded or unrecognized sections. These formats are designed for easy 4-byte alignment. Be sure your compiler/linker doesn't insert any extra padding between values. If necessary, use your compiler's "packed" directive.
2. All values have big-endian byte order. Your compiler may provide conversion functions such as htonl, htons, ntohl, ntohs, however those assume integers so you'll need to be very careful with floats.
3. u8, u16, u32 means unsigned integers of 8, 16, 32 bits.
   s8, s16, s32 means signed integers of 8, 16, 32 bits.
   f32 means IEEE-754 32-bit floating point.
4. All packets are UDP/IP datagrams

15.3 Port Numbers

```
Bathymetry data port          = gui.Baseport + 0
TruePix data port             = tpd.Baseport + 1
Device status port            = gui.Baseport + 2
Acoustic Image data port      = gui.Baseport + 3
Water Column data port        = wcd.Baseport + 5
Snippets data port            = tpd.Baseport + 6
```

15.4 Type Definitions

```
typedef unsigned char  u8;
typedef unsigned short u16;
typedef unsigned int   u32;
typedef signed char    s8;
typedef signed short   s16;
typedef signed int     s32;
typedef float          f32;
```

15.5 Ethernet Data Rates

Bathymetry: ≈ 800 kb/s max (bathy data is sent twice, to GUI and data acquisition computer)
TruePix: ≈ 5.5 Mb/s (magnitude + angle) max
≈ 3.5 Mb/s (magnitude) max
Water Column: ≈ 560 Mb/s (magnitude + phase) max
≈280 Mb/s (magnitude) max
Snippets: ≈ 11 Mb/s max

Where Mb/s = megabits per second.

15.6 Bathymetry Packet Format

```
// *** BEGIN PACKET: BATHY DATA FORMAT 0 ***

 u32  PacketName;              // 'BTH0'
 u32  PacketSize;             // [bytes] size of this entire packet
 u32  DataStreamID;            // reserved for future use


 // section H0: header

 u16  H0_SectionName;          // 'H0'
 u16  H0_SectionSize;          // [bytes] size of this entire section
 u8   H0_ModelNumber[12];      // example "2024", unused chars are nulls
 u8   H0_SerialNumber[12];     // example "100017", unused chars are nulls
 u32  H0_TimeSeconds;          // [seconds] ping time relative to 0000 hours 1-Jan-1970, integer part
 u32  H0_TimeNanoseconds;      // [nanoseconds] ping time relative to 0000 hours 1-Jan-1970, fraction part
 u32  H0_PingNumber;           // pings since power-up or reboot
 f32  H0_PingPeriod;           // [seconds] time between most recent two pings
 f32  H0_SoundSpeed;           // [meters per second]
 f32  H0_Frequency;            // [hertz] sonar center frequency
 f32  H0_TxPower;              // [dB re 1 uPa at 1 meter]
 f32  H0_TxPulseWidth;         // [seconds]
 f32  H0_TxBeamwidthVert;      // [radians]
 f32  H0_TxBeamwidthHoriz;     // [radians]
 f32  H0_TxSteeringVert;       // [radians]
 f32  H0_TxSteeringHoriz;      // [radians]
 u16  H0_2026ProjTemp;         // [hundredths of a degree Kelvin] 2026 projector temperature (divide value by 100, subtract 273.15 to get °C)
 s16  H0_VTX+Offset;           // [hundredths of a dB] transmit voltage offset at time of ping (divide value by 100 to get dB)
 f32  H0_RxBandwidth;          // [hertz]
 f32  H0_RxSampleRate;         // [hertz] sample rate of data acquisition and signal processing
 f32  H0_RxRange;              // [meters] sonar range setting
 f32  H0_RxGain;               // [multiply by two for relative dB]
 f32  H0_RxSpreading;          // [dB (times log range in meters)]
 f32  H0_RxAbsorption;         // [dB per kilometer]
 f32  H0_RxMountTilt;          // [radians]
 u32  H0_RxMiscInfo;           // reserved for future use
 u16  H0_reserved;             // reserved for future use
 u16  H0_Points;               // number of bathy points
```

// section R0: 16-bit bathy point ranges

```
u16  R0_SectionName;           // 'R0'
u16  R0_SectionSize;           // [bytes] size of this entire section
f32  R0_ScalingFactor;
u16  R0_Range[H0_Points];      // [seconds two-way] = R0_Range * R0_ScalingFactor
u16  R0_unused[H0_Points & 1];   // ensure 32-bit section size
```

// section A0: bathy point angles, equally-spaced (present only during "equi-angle" spacing mode)

```
u16  A0_SectionName;           // 'A0'
u16  A0_SectionSize;           // [bytes] size of this entire section
f32  A0_AngleFirst;            // [radians] angle of first (port side) bathy point, relative to array centerline, AngleFirst < AngleLast
f32  A0_AngleLast;             // [radians] angle of last (starboard side) bathy point
f32  A0_MoreInfo_0;            // 0 (reserved for future use)
f32 A0_MoreInfo_1              //Z-offset, proj [metres]
f32 A0_MoreInfo_2;             //Y-offset, proj [metres]
f32 A0_MoreInfo_3;             //X-offset, proj [metres]
f32 A0_MoreInfo_4;             //0 (reserved for future use)
f32 A0_MoreInfo_5;             //0 (reserved for future use)
```

// section A2: 16-bit bathy point angles, arbitrarily-spaced (present only during "equi-distant" spacing mode)

```
u16  A2_SectionName;           // 'A2'
u16  A2_SectionSize;           // [bytes] size of this entire section
f32  A2_AngleFirst;            // [radians] angle of first (port side) bathy point, relative to array centerline, AngleFirst < AngleLast
f32  A2_ScalingFactor;
 f32  A0_MoreInfo_0;           // 0 (reserved for future use)
f32 A0_MoreInfo_1              //Z-offset, proj [metres]
f32 A0_MoreInfo_2;             //Y-offset, proj [metres]
f32 A0_MoreInfo_3;             //X-offset, proj [metres]
f32 A0_MoreInfo_4;             //0 (reserved for future use)
f32 A0_MoreInfo_5;             //0 (reserved for future use)
 u16  A2_AngleStep[H0_Points];    // [radians] angle[n] = A2_AngleFirst + (32-bit sum of A2_AngleStep[0] through A2_AngleStep[n]) * A2_ScalingFactor
u16  A2_unused[H0_Points & 1];   // ensure 32-bit section size
```

// section I1: 16-bit bathy intensity (present only if enabled)

```
u16  I1_SectionName;           // 'I1'
u16  I1_SectionSize;           // [bytes] size of this entire section
f32  I1_ScalingFactor;
u16  I1_Intensity[H0_Points];    // [micropascals] intensity[n] = I1_Intensity[n]) * I1_ScalingFactor
u16  I1_unused[H0_Points & 1];   // ensure 32-bit section size
```

```
// section G0: simple straight-line depth gates

u16  G0_SectionName;          // 'G0'
u16  G0_SectionSize;          // [bytes] size of this entire section
f32  G0_DepthGateMin;         // [seconds two-way]
f32  G0_DepthGateMax;         // [seconds two-way]
f32  G0_DepthGateSlope;       // [radians]

// section G1: 8-bit gate positions, arbitrary paths (present only during "verbose" gate description mode)

u16  G1_SectionName;          // 'G1'
u16  G1_SectionSize;          // [bytes] size of this entire section
f32  G1_ScalingFactor;
struct
{
  u8   RangeMin;              // [seconds two-way] = RangeMin * G1_ScalingFactor
  u8   RangeMax;              // [seconds two-way] = RangeMax * G1_ScalingFactor
}   G1_Gate[H0_Points];
u16  G1_unused[H0_Points & 1];   // ensure 32-bit section size

// section Q0: 4-bit quality flags

u16  Q0_SectionName;          // 'Q0' quality, 4-bit
u16  Q0_SectionSize;          // [bytes] size of this entire section
u32  Q0_Quality[(H0_Points+7)/8]; // 8 groups of 4 flags bits (phase detect, magnitude detect, reserved, reserved), packed left-to-right

// *** END PACKET: BATHY FORMAT 0 ***
```

15.7 Snippet Format

```
// *** BEGIN PACKET: SNIPPET DATA FORMAT 0 ***

 u32  PacketName;              // 'SNI0'
 u32  PacketSize;             // may be zero in UDP, otherwise: [bytes] size of this entire packet
 u32  DataStreamID;            // reserved for future use

 // section H0: header (present only in first snippet packet of each ping)

 u16  H0_SectionName;          // 'H0'
 u16  H0_SectionSize;          // [bytes] size of this entire section
 u8   H0_ModelNumber[12];      // example "2024", unused chars are nulls
 u8   H0_SerialNumber[12];     // example "100017", unused chars are nulls
 u32  H0_TimeSeconds;          // [seconds] ping time relative to 0000 hours 1-Jan-1970, integer part
 u32  H0_TimeNanoseconds;      // [nanoseconds] ping time relative to 0000 hours 1-Jan-1970, fraction part
 u32  H0_PingNumber;           // pings since power-up or reboot
 f32  H0_PingPeriod;           // [seconds] time between most recent two pings
 f32  H0_SoundSpeed;           // [meters per second]
 f32  H0_Frequency;            // [hertz] sonar center frequency
 f32  H0_TxPower;              // [dB re 1 uPa at 1 meter]
 f32  H0_TxPulseWidth;         // [seconds]
 f32  H0_TxBeamwidthVert;      // [radians]
 f32  H0_TxBeamwidthHoriz;     // [radians]
 f32  H0_TxSteeringVert;       // [radians]
 f32  H0_TxSteeringHoriz;      // [radians]
 u16  H0_2026ProjTemp;         // [hundredths of a degree Kelvin] 2026 projector temperature (divide value by 100, subtract 273.15 to get °C)
 s16  H0_VTX+Offset;           // [hundredths of a dB] transmit voltage offset at time of ping (divide value by 100 to get dB)
 f32  H0_RxBandwidth;          // [hertz]
 f32  H0_RxSampleRate;         // [hertz] sample rate of data acquisition and signal processing
 f32  H0_RxRange;              // [meters] sonar range setting
 f32  H0_RxGain;               // [multiply by two for relative dB]
 f32  H0_RxSpreading;          // [dB (times log range in meters)]
 f32  H0_RxAbsorption;         // [dB per kilometer]
 f32  H0_RxMountTilt;          // [radians]
 u32  H0_RxMiscInfo;           // reserved for future use
 u16  H0_reserved;             // reserved for future use
 u16  H0_Snippets;             // number of snippets
  f32  A0_MoreInfo_0;          // 0 (reserved for future use)
  f32  A0_MoreInfo_1           //Z-offset, proj [metres]
  f32  A0_MoreInfo_2;          //Y-offset, proj [metres]
  f32  A0_MoreInfo_3;          //X-offset, proj [metres]
  f32  A0_MoreInfo_4;          //0 (reserved for future use)
  f32  A0_MoreInfo_5;          //0 (reserved for future use)
```

```
// section S1: 16-bit snippet data (for network efficiency packet may contain several of these sections) (supports snippets up to 32K samples by fragmenting
// at the IP level rather than by the application like 81xx)

u16  S1_SectionName;           // 'S1'
u16  S1_SectionSize;          // [bytes] size of this entire section
u32  S1_PingNumber;           // pings since power-up or reboot
u16  S1_SnippetNumber;        // snippet number, 0 to H0_Snippets-1
u16  S1_Samples;              // number of samples in this snippet, sample rate is H0_RxSampleRate
u32  S1_FirstSample;          // first sample of this snippet relative to zero range, sample rate is H0_RxSampleRate
f32  S1_Angle;                // [radians] angle of this snippet, relative to array centerline
f32  S1_ScalingFactorFirst;   // scaling factor at start of snippet, 0=ignore, use linear interpolation to get other values
f32  S1_ScalingFactorLast;    // scaling factor at end of snippet, 0=ignore
u32  S1_reserved;             // reserved for future use
u16  S1_Magnitude[S1_Samples];   // [micropascals] = S1_Magnitude[n] * (linear interpolate between S1_ScalingFactorFirst and S1_ScalingFactorLast)
u16  S1_unused[S1_Samples & 1];  // ensure 32-bit section size
```

// *** END PACKET: SNIPPET DATA FORMAT 0 ***

15.8 Water Column (WC) Data Format

```
// *** BEGIN PACKET: WATER COLUMN (WC) DATA FORMAT 0 ***

 // The water column data contains real-time beamformer 16-bit magnitude data
 // (beam amplitude) and optional 16-bit split-array phase data (intra-beam
 // direction). Maximum data rate is about 70 megabytes per second (assuming
 // 256 beams, 68.4 kHz sample rate, and phase data enabled). The sample rate
 // (and signal bandwidth) varies with transmit pulse width and range setting.
 // Maximum ping data size is about 32 megabytes (assuming 256 beams of 32768
 // samples, and phase data enabled), but max size may change in the future.
 // The number of beamformed data samples normally extends somewhat further
 // than the user's range setting.
 //
 // When the operator enables water column mode, each sonar ping outputs
 // numerous 'WCD0' packets containing: one H0 header section, one A1 beam
 // angle section, and many M1 or M2 data sections. The section order may
 // change in the future, so plan for that in your data acquisition.
 //
 // Each M1 or M2 section contains a subset of the ping data. Its header
 // indicates its size position to help you assemble the full ping array.
 //
 // You may wish to detect missing M1 or M2 data sections (perhaps a lost
  //  DP packet), and then fill the gap with zeros or perhaps data from the
 // previous ping (to reduce visual disturbances), and then increment an
 // error counter for network health monitoring purposes.
 //
 // The water column data is basically in polar coordinates, so you may
 // wish to geometrically warp it into the familiar wedge shape for display.
 // Consider using OpenGL or Direct3D texture mapping.

 u32  PacketName;              // 'WCD0'
 u32  PacketSize;             // [bytes] size of this entire packet
 u32  DataStreamID;            // reserved for future use
```

// section H0: header (only one per ping)

```
u16  H0_SectionName;           // 'H0'
u16  H0_SectionSize;          // [bytes] size of this entire section
  u8   H0_ModelNumber[12];       // example "2024", unused chars are nulls
u8   H0_SerialNumber[12];      // example "100017", unused chars are nulls
u32  H0_TimeSeconds;          // [seconds] ping time relative to 0000 hours 1-Jan-1970, integer part
u32  H0_TimeNanoseconds;       // [nanoseconds] ping time relative to 0000 hours 1-Jan-1970, fraction part
u32  H0_PingNumber;           // pings since power-up or reboot
  f32  H0_PingPeriod;           // [seconds] time between most recent two pings
f32  H0_SoundSpeed;           // [meters per second]
f32  H0_Frequency;            // [hertz] sonar center frequency
f32  H0_TxPower;              // [dB re 1 uPa at 1 meter]
f32  H0_TxPulseWidth;         // [seconds]
f32  H0_TxBeamwidthVert;      // [radians]
f32  H0_TxBeamwidthHoriz;     // [radians]
f32  H0_TxSteeringVert;       // [radians]
f32  H0_TxSteeringHoriz;      // [radians]
u16  H0_TxMiscInfo;           // reserved for future use
s16  H0_VTX+Offset;           // [hundredths of a dB] transmit voltage offset at time of ping (divide value by 100 to get dB)
f32  H0_RxBandwidth;          // [hertz]
f32  H0_RxSampleRate;         // [hertz] sample rate of data acquisition and signal processing
f32  H0_RxRange;              // [meters] sonar range setting
f32  H0_RxGain;               // [multiply by two for relative dB]
f32  H0_RxSpreading;          // [dB (times log range in meters)]
f32  H0_RxAbsorption;         // [dB per kilometer]
f32  H0_RxMountTilt;          // [radians]
u32  H0_RxMiscInfo;           // reserved for future use
u16  H0_reserved;             // reserved for future use
u16  H0_Beams;                // number of beams
```

// section A1: float beam angles, arbitrarily-spaced (only one per ping)

```
 u16  A1_SectionName;          // 'A1'
u16  A1_SectionSize;          // [bytes] size of this entire section
f32  A0_MoreInfo_0;           // 0 (reserved for future use)
f32  A0_MoreInfo_1            //Z-offset, proj [metres]
f32  A0_MoreInfo_2;           //Y-offset, proj [metres]
f32  A0_MoreInfo_3;           //X-offset, proj [metres]
f32  A0_MoreInfo_4;           //0 (reserved for future use)
f32  A0_MoreInfo_5;          //0 (reserved for future use)
f32  A1_BeamAngle[H0_Beams];     // [radians] angle of beam relative to array centerline, ordered from port to starboard, first angle < last angle
```

// section M1: 16-bit magnitude data (present only during "magnitude-only" water column data mode, many per ping, you assemble them into complete ping data)

```
u16  M1_SectionName;          // 'M1'
u16  M1_SectionSize;          // [bytes] size of this entire section
u32  M1_PingNumber;           // pings since power-up or reboot
```

```
f32  M1_ScalingFactor;          // reserved for future use
u32  M1_TotalSamples;           // range samples in entire ping, sample rate is H0_RxSampleRate
u32  M1_FirstSample;            // first sample of this section
u16  M1_Samples;                // number of samples in this section
u16  M1_TotalBeams;             // beams (always a multiple of 2) (typically columns in your memory buffer)
u16  M1_FirstBeam;              // first beam of this section (always a multiple of 2)
u16  M1_Beams;                  // number of beams in this section (always a multiple of 2)
u32  M1_reserved0;              // reserved for future use
 u32  M1_reserved1;             // reserved for future use
struct

{
  u16  magnitude;               // values 0 to 65535 map non-linearly (due to TVG scaling and possible gain compression) to signal amplitude
} M1_Data[M1_Beams][M1_Samples];  // magnitude data (typical example: 256 beams each containing 36 two-byte structs, 16 kilobytes)

// section M2: 16-bit magnitude and phase data (present only during "magnitude and phase" water column data mode, many per ping, you assemble them into
// complete ping data)

u16  M2_SectionName;            // 'M2'
u16  M2_SectionSize;            // [bytes] size of this entire section
u32  M2_PingNumber;             // pings since power-up or reboot
f32  M2_ScalingFactor;          // reserved for future use
u32  M2_TotalSamples;           // range samples in entire ping, sample rate is H0_RxSampleRate
u32  M2_FirstSample;            // first sample of this section
u16  M2_Samples;                // number of samples in this section
u16  M2_TotalBeams;             // beams (always a multiple of 2) (typically columns in your memory buffer)
u16  M2_FirstBeam;              // first beam of this section (always a multiple of 2)
u16  M2_Beams;                  // number of beams in this section (always a multiple of 2)
u32  M2_reserved0;              // reserved for future use
u32  M2_reserved1;              // reserved for future use
struct
{
  u16  magnitude;               // values 0 to 65535 map non-linearly (due to TVG scaling and possible gain compression) to signal amplitude
  s16  phase;                   // values -32768 to +32767 map non-linearly (due to complex transfer function) to target angle within the beamwidth
} M2_Data[M2_Beams][M2_Samples];  // magnitude and phase data (typical example: 256 beams each containing 36 four-byte structs, 36 kilobytes)


// *** END PACKET: WATER COLUMN (WC) DATA FORMAT 0 ***
```

15.9 Acoustic Image (AI) Data Format

```
// *** BEGIN PACKET: ACOUSTIC IMAGE (AI) DATA FORMAT 0 ***
 // The acoustic image data contains real-time beamformer 8-bit magnitude data
 // (beam amplitude) that has been scaled to 8-bits by a user-selected
 // brightness value, and compressed in range by an adjustable amount to
 // reduce network bandwidth and processing. The data is called "samples"
 // before compression and "bins" after compression. For example, 7200 samples
 // of beamformer data (M0_TotalSamples) may be compressed to 600 bins
 // (M0_TotalBins). The number of beamformed data samples normally extends
 // somewhat further than the user's range setting. The AIH0 sonar command
 // sets an upper limit to the number of compressed output bins. It's not a
 // precise compression factor, so the number of bins is usually somewhat less
 // than the AIH0 value. The maximum data rate with no compression is about
 // 17.5 megabytes per second (assuming 256 beams).
 //
 // When the operator enables acoustic image mode, each sonar ping outputs
 // numerous 'AID0' packets containing: one H0 header section, one A1 beam
 // angle section, and many M0 data sections. The section order may change in
 // the future, so plan for that in your data acquisition.
 //
 // Each M0 section contains a subset of the ping data. Its header indicates
 // its size position to help you assemble the full ping array.
 //
 // You may wish to detect missing M0 data sections (perhaps a lost UDP
 // packet), and then fill the gap with zeros or perhaps data from the
 // previous ping (to reduce visual disturbances), and then increment an error
 // counter for network health monitoring purposes.
 //
 // The acoustic image data is basically in polar coordinates, so you may wish
 // to geometrically warp it into the familiar wedge shape for display.
 // Consider using OpenGL or Direct3D texture mapping.

 u32  PacketName;              // 'AID0'
 u32  PacketSize;             // [bytes] size of this entire packet
 u32  DataStreamID;            // reserved for future use

 // section H0: header (only one per ping)

 u16  H0_SectionName;          // 'H0'
```

```
u16  H0_SectionSize;           // [bytes] size of this entire section
u8   H0_ModelNumber[12];       // example "2024", unused chars are nulls
u8   H0_SerialNumber[12];      // example "100017", unused chars are nulls
u32  H0_TimeSeconds;           // [seconds] ping time relative to 0000 hours 1-Jan-1970, integer part
u32  H0_TimeNanoseconds;       // [nanoseconds] ping time relative to 0000 hours 1-Jan-1970, fraction part
u32  H0_PingNumber;            // pings since power-up or reboot
f32  H0_PingPeriod;            // [seconds] time between most recent two pings
f32  H0_SoundSpeed;            // [meters per second]
f32  H0_Frequency;             // [hertz] sonar center frequency
f32  H0_TxPower;               // [dB re 1 uPa at 1 meter]
f32  H0_TxPulseWidth;          // [seconds]
f32  H0_TxBeamwidthVert;       // [radians]
f32  H0_TxBeamwidthHoriz;      // [radians]
f32  H0_TxSteeringVert;        // [radians]
f32  H0_TxSteeringHoriz;       // [radians]
u16  H0_2026ProjTemp;          // [hundredths of a degree Kelvin] 2026 projector temperature (divide value by 100, subtract 273.15 to get °C)
s16  H0_VTX+Offset;            // [hundredths of a dB] transmit voltage offset at time of ping (divide value by 100 to get dB)
f32  H0_RxBandwidth;           // [hertz]
f32  H0_RxSampleRate;          // [hertz] sample rate of data acquisition and signal processing
f32  H0_RxRange;               // [meters]
f32  H0_RxGain;                // [multiply by two for relative dB]
f32  H0_RxSpreading;           // [dB (times log range in meters)]
f32  H0_RxAbsorption;          // [dB per kilometer]
f32  H0_RxMountTilt;           // [radians]
u32  H0_RxMiscInfo;            // reserved for future use
u16  H0_reserved;              // reserved for future use
u16  H0_Beams;                 // number of beams beams
```

// section A1: float beam angles, arbitrarily-spaced (only one per ping)

```
u16  A1_SectionName;           // 'A1'
u16  A1_SectionSize;           // [bytes] size of this entire section
f32  A0_MoreInfo_0;            // 0 (reserved for future use)
f32  A0_MoreInfo_1             //Z-offset, proj [metres]
f32  A0_MoreInfo_2;            //Y-offset, proj [metres]
f32  A0_MoreInfo_3;            //X-offset, proj [metres]
f32  A0_MoreInfo_4;            //0 (reserved for future use)
f32  A0_MoreInfo_5;            //0 (reserved for future use)  f32  A1_BeamAngle[H0_Beams];     // [radians] angle of beam relative to array centerline, ordered from port to starboard, first angle < last angle
```

// section M0: 8-bit magnitude data (many per ping, you assemble them into complete ping data)

```
u16  M0_SectionName;           // 'M0'
u16  M0_SectionSize;           // [bytes] size of this entire section
u32  M0_PingNumber;            // pings since power-up or reboot
f32  M0_ScalingFactor;         // reserved for future use
u32  M0_TotalSamples;          // range samples (before compression) in entire ping, sample rate is H0_RxSampleRate
u32  M0_TotalBins;             // range bins (after compression) in entire ping (M0_TotalBins <= M0_TotalSamples)
u32  M0_FirstBin;              // first bin of this section
u16  M0_Bins;                  // number of bins in this section
u16  M0_TotalBeams;            // beams (always a multiple of 4) (typically columns in your memory buffer)
```

```
 u16  M0_FirstBeam;             // first beam of this section (always a multiple of 4)
 u16  M0_Beams;                 // number of beams in this section (always a multiple of 4)
 u32  M0_reserved;              // reserved for future use
 struct
 {    u8  magnitude;            // values 0 to 255 map non-linearly (due to TVG scaling and possible gain compression) to signal amplitude
 } M0_Data[M0_Beams][M0_Bins];    // magnitude data (typical example: 256 beams each containing 21 one-byte structs, 5376 bytes)

// *** END PACKET: ACOUSTIC IMAGE (AI) DATA FORMAT 0 ***
```

15.10 TruePix™ Data Format

```
// *** BEGIN TRUEPIX DATA FORMAT 0 ***
 // TruePix is like sidescan with 3D relief. Each sonar ping produces a port
 // and starboard time-series of data samples at the sonar's sample rate. Each
 // sample contains the signal's magnitude (like sidescan) and across-track
 // target direction angle (like bathymetry). After collecting many pings of
 // data along a survey line, you now have a large array of data points with
 // range, direction, and brightness. Apply noise reduction, and render the
 // data as a textured 3D surface.
 //
 // Two data formats are available: D0 provides magnitudes only, D1 provides
 // magnitudes and direction angles. The GUI allows the user to choose the
 // desired format.
 //
 // The sonar generates one TruePix data set per ping. Each data set is
 // usually split into multiple UDP packets. The D0 or D1 header includes
 // FirstSample and Samples values to help you reassemble the full data set.
 //
 // Someday you may be able to convert the 16-bit magnitude values to
 // micropascals by applying a to-be-determined function involving the sample
 // number and the MagnitudeScaling[] coefficients, but this conversion is not
 // yet supported so these coefficients are zero. You can convert the
 // direction angles from 16-bit values to radians by multiplying by
 // AngleScalingFactor.

 u32  PacketName;               // 'TPX0'
 u32  PacketSize;               // may be zero in UDP, otherwise: [bytes] size of this entire packet
 u32  DataStreamID;             // reserved for future use

 // section H0: header (present only in first packet of each ping)
 u16  H0_SectionName;           // 'H0'
 u16  H0_SectionSize;           // [bytes] size of this entire section
```

```
u8   H0_ModelNumber[12];        // example "2024", unused chars are nulls
u8   H0_SerialNumber[12];       // example "100017", unused chars are nulls
u32  H0_TimeSeconds;            // [seconds] ping time relative to 0000 hours 1-Jan-1970, integer part
u32  H0_TimeNanoseconds;        // [nanoseconds] ping time relative to 0000 hours 1-Jan-1970, fraction part
u32  H0_PingNumber;             // pings since power-up or reboot
f32  H0_PingPeriod;             // [seconds] time between most recent two pings
f32  H0_SoundSpeed;             // [meters per second]
f32  H0_Frequency;              // [hertz] sonar center frequency
f32  H0_TxPower;                // [dB re 1 uPa at 1 meter]
f32  H0_TxPulseWidth;           // [seconds]
f32  H0_TxBeamwidthVert;        // [radians]
f32  H0_TxBeamwidthHoriz;       // [radians]
f32  H0_TxSteeringVert;         // [radians]
f32  H0_TxSteeringHoriz;        // [radians]
u16  H0_2026ProjTemp;           // [hundredths of a degree Kelvin] 2026 projector temperature (divide value by 100, subtract 273.15 to get °C)
s16  H0_VTX+Offset;             // [hundredths of a dB] transmit voltage offset at time of ping (divide value by 100 to get dB)
f32  H0_RxBandwidth;            // [hertz]
f32  H0_RxSampleRate;           // [hertz] sample rate of data acquisition and signal processing
f32  H0_RxRange;                // user setting [meters]
f32  H0_RxGain;                 // user setting [multiply by 2 for dB]
f32  H0_RxSpreading;            // [dB (times log range in meters)]
f32  H0_RxAbsorption;           // [dB per kilometer]
f32  H0_RxMountTilt;            // [radians]
u32  H0_RxMiscInfo;             // reserved for future use
u32  H0_reserved;               // reserved for future use
f32  A0_MoreInfo_0;             // 0 (reserved for future use)
f32  A0_MoreInfo_1              //Z-offset, proj [metres]
f32  A0_MoreInfo_2;             //Y-offset, proj [metres]
f32  A0_MoreInfo_3;             //X-offset, proj [metres]
f32  A0_MoreInfo_4;             //0 (reserved for future use)
f32  A0_MoreInfo_5;             //0 (reserved for future use)

// section D0: 16-bit magnitude data (present only during "magnitude only" mode)
u16  D0_SectionName;            // 'D0'
u16  D0_SectionSize;            // [bytes] size of this entire section
u32  D0_PingNumber;             // pings since power-up or reboot
u32  D0_TotalSamples;           // number of samples in entire time series (sample rate is H0_RxSampleRate)
u32  D0_FirstSample;            // first sample of this section relative to zero range
u16  D0_Samples;                // number of samples in this section
u16  D0_reserved;               // reserved for future use
f32  D0_MagnitudeScaling[8];    // to be determined, 0=ignore
struct
{
  u16 PortMagnitude;            // [micropascals] = PortMagnitude * (tbd function of sample number and D0_MagnitudeScaling[8])
  u16 StbdMagnitude;            // similar but starboard side
} D0_Data[D0_Samples];

// section D1: 16-bit magnitude and direction data (present only during "magnitude+direction" mode)

u16  D1_SectionName;            // 'D1'
u16  D1_SectionSize;            // [bytes] size of this entire section
```

```
 u32  D1_PingNumber;            // pings since power-up or reboot
 u32  D1_TotalSamples;          // number of samples in entire time series (sample rate is H0_RxSampleRate)
 u32  D1_FirstSample;           // first sample of this section relative to zero range
 u16  D1_Samples;               // number of samples in this section
 u16  D1_reserved;              // reserved for future use
 f32  D1_MagnitudeScaling[8];     // to be determined, 0=ignore
 f32  D1_AngleScalingFactor;
 struct
 {
  u16 PortMagnitude;            // [micropascals] = PortMagnitude * (tbd function of sample number and D1_MagnitudeScaling[8])
  s16 PortAngle;               // [radians from array centerline (positive towards starboard)] = PortAngle * D1_AngleScalingFactor
  u16 StbdMagnitude;            // similar but starboard side
  s16 StbdAngle;               // similar but starboard side
 } D1_Data[D1_Samples];

// *** END TRUEPIX DATA FORMAT 0 ***
```

15.11 Head Status Format

```
// *** BEGIN PACKET: HEAD STATUS DATA FORMAT 0 ***
 // Head Status data reports the status of the sonar head. This data is
 // useful for troubleshooting. Data is sent to gui baseport + 2.
 //
 // Each section name consists of 4 characters. The fourth character
 // indicates the number of 32-bit words following each section name.
 // The forth character can be 1-9, A-Z; allowing up to 35 32-bit words.
 // The number of words in each section may change at a later date. Be
 // sure your program can parse the number of words.
 // The order of the sections is not fixed.

 u32  PacketName;               // 'STH0'
 u32  PacketSize;              // [bytes] size of this entire packet
 u32  DataStreamID;             // reserved for future use

 // section SER3: serial number

 u32  SER3_SectionName;        // 'SER3'
 u32  serial_number[3];        //   example "100117", unused chars are nuls

 // section PRT3: part number

 u32  PRT3_SectionName;        // 'PRT3'
 u32  part_number[3];          //   example "15000001", unused chars are nuls
```

```
// section MDL3: model number

u32  MDL3_SectionName;        // 'MDL3'
u32  model_number[3];         //   example "2024", unused chars are nuls

// section FWV6: main controller firmware version

u32  'FWV6';                  // main ctrl firmware version string
u32  version.i[6];            //   example "19-Dec-2011-17:19:29", unused chars are nuls

// section FWT6: internal transmitter firmware version

u32  FWT6_SectionName;        // 'FWT6'
u32  tinytx.i[6];             //   example "16-Aug-2011-17:19:29", unused chars are nuls

// section PRJ9: projector

u32  PRJ9_SectionName;        // 'PRJ9'
u32  serial_number[3]         //   example "800456", unused chars are nuls
u32  part_number[3];          //   example "15000004", unused chars are nuls
u32  model_number[3];         //   example "1004", unused chars are nuls


// section OPT1: option settings

u32  OPT1_SectionName         // 'OPT1'
u32  options                  //   truepix_snippets[0:0]       0=off,  1=on
                              //   depth_rating[1:1]           0=100m, 1=3km
                              //   forward_looking[2:2]        0=off,  1=on
                              //   water_column[3:3]           0=off,  1=on
                              //   ultra-high resolution[4:4]  0=off,  1=on
                              //   water column rx chan [5:5]  0=off,  1=on
                              //   2026 low freq option [6:6]  0=off,  1=90 & 100kHz enabled, 2026 only

// section CFG8: configuration parameters
u32  CFG8 SectionName;        // 'CFG8'
f32  pdepth_cutout;           //   [meters] pressure cutout depth (no bathy detect below this depth)
f32  pdepth_export_depth;     //   [meters] pressure trigger for export control (limits lowest freq and range)
f32  pdepth_export_bathyrange; //   [meters] max bathy range when pressure trigger is exceeded
f32  pdepth_export_minfreq;   //   [Hz] min freq when pressure trigger is exceeded
f32  option_lowfreq;          //   [Hz] low-frequency option
f32  pdepth_export_bathydepth; //   [meters] max bathy depth allowed
f32  spare0;                  //   spare
f32  spare1;                  //   spare

// section OPR4: operational parameters so GUI can configure its menus
u32  OPR4 SectionName;        // 'OPR4'
f32  opr_minfreq;             //   [Hz] current minimum selectable frequency (zero means ignore me and use GUI default)
```

```
f32  opr_maxfreq;           //  [Hz] current maximum selectable frequency (zero means ignore me and use 400kHz default)
f32  spare0;                //  spare
f32  spare1;                //  spare

// section SENa: sensor data received from SIM

u32  SENa_SectionName;      // 'SENa'
u32  gps.time.sec;          //  [seconds]unix time
u32  gps.time.nsec;         //  [seconds = gps.time.nsec/(2^32)] unix time
f32  sensor.pitch;          //  [radians] mru pitch
f32  sensor.roll;           //  [radians] mru roll
f32  sensor.heave;          //  [meters]  mru heave
f32  sensor.heading;        //  heading (not implemented)
f32  sensor.velocity;       //  [m/s] sound velocity
f32  sensor.pdepth.uncal;   //  [meters] depth uncalibrated
f32  sensor.pdepth.cal;     //  [meters] depth calibrated
f32  sensor.fpgatemp;       //  [°C] FPGA temperature

// section ADC3: a/d converter

u32  ADC3_SectionName;      // 'ADC3'
f32  adc.chan0;             //  [volts] 48VDC power supply voltage
f32  adc.chan1;             //  [amperes] 48V current
f32  adc.chan8;             //  [volts] transmitter power supply voltage



// section ETH6; ethernet registers

u32  ETH6_SectionName;      // 'ETH6'
u32  ethernet.speed;        //  [megabits/sec] link connect speed
u32  erxpackets;            //  [counts] ethernet receive packets
u32  etxpackets;            //  [counts] ethernet transmit packets
u32  erxoverflows;          //  [counts] ethernet receive buffer overflows
u8   mac.addr[8]            //  mac address, use last 6 bytes, first 2 bytes are not used

// section TIM2; timers

u32  TIM2_SectionName;      // 'TIM2'
f32  time.check;            //  [seconds] head to sim roundtrip time response (must be less than 3ms for dual head or ext. trigger modes)
f32  time.spare;            //  [seconds] spare// *** END PACKET: HEAD STATUS DATA FORMAT 0 ***
```

15.12 SIM Status Data Format

```
// *** BEGIN PACKET: SIM STATUS DATA FORMAT 0 ***
 // SIM Status data reports misc info from the SIM box. This data is
 // useful for troubleshooting. Data is sent to gui baseport+2.
 //
 // Each section name consists of 4 characters. The fourth character
 // indicates the number of 32-bit words following each section name.
 // The forth character can be 1-9, A-Z; allowing up to 35 32-bit words.
 // The number of words in each section may change at a later date. Be
 // sure your program can parse the number of words.
 // The order of the sections is not fixed.

 u32  PacketName;            // 'STS0'
 u32  PacketSize;           // [bytes] size of this entire packet
 u32  DataStreamID;          // reserved for future use

 // section SER3: serial number

 u32  SER3_SectionName;       // 'SER3'
 u32  serial_number[3];       //   example "100117", unused chars are nulls

 // section PRT3: part number

 u32  PRT3_SectionName;       // 'PRT3'
 u32  part_number[3];         //   example "15000002", unused chars are nulls

 // section MDL3: model number

 u32  MDL3_SectionName;       // 'MDL3'
 u32  model_number[3];        //   example "2024", unused chars are nulls

 // section FWV6: firmware version

 u32  FWV6_SectionName;       // 'FWV6'
 u32  version[6];             //   example "15-Dec-2011-14:00:42", unused chars are nulls
```

```
// section LED1: SIM front panel LED status

u32  LED1_SectionName;        // 'LED1'
u32  led_status;              //  [00=off 01=undef 10=bad 11=good] flags for status LEDs
                              //  gps[1:0]
                              //  motion[3:2]
                              //  heading[5:4], not implemented
                              //  svp[7:6]
                              //  alt-gps[9:8], not implemented
                              //  alt-motion[11:10], not implemented
                              //  alt-heading[13:12], not implemented
                              //  alt-svp[15:14], not implemented
                              //  pps[17:16]
                              //  sync in[19:18]
                              //  sync out[21:20]
                              //  head on[23:22]
                              //  reserved[31:24]

// section SEN7: RS232 sensor values

u32  SEN7_SectonName;         // 'SEN7'
u32  gps.time.sec;            //  [seconds] unix time
u32  gps.time.nsec;           //  [seconds = gps.time.nsec/(2^32)] unix time
f32  mru.pitch;               //  [radians] mru pitch value
f32  mru.roll;                //  [radians] mru roll value
f32  mru.heave;               //  [meters]  mru heave
f32  0.0;                     //  heading (not implemented)
f32  svp.velocity;            //  [m/s] sound velocity

// section ADC2: a/d converter

u32  ADC2_SectonName;         // 'ADC2'
f32  adc.chan0;               //  [volts] 48VDC power supply voltage
f32  adc.chan1;               //  [amperes] 48V current to head

// section ETH6: ethernet registers

u32  ETH6_SectonName;         // 'ETH6'
u32  ethernet.speed;          //  [megabits/sec] link speed
u32  erxpackets;              //  [counts] ethernet receive packets
u32  etxpackets;              //  [counts] ethernet transmit packets
u32  erxoverflows;            //  [counts] ethernet receive buffer overflows
u8   mac.addr[8]              //  mac address, use last 6 bytes, first 2 bytes are not used

// *** END PACKET: SIM STATUS DATA FORMAT 0 ***
```

15.13 Device Status Format

The device status packet contains the ConfigID number that was sent to the sonar head and SIM during IP configuration. This packet contains no survey information and is ignored for data collection purposes. The R2DS packet is sent from the sonar head and SIM once per second to the sonar control program IP address. The ConfigID received from the sonar head and SIM should be compared with the ConfigID number sent to the sonar head and SIM during IP configuration. If there is a mismatch, the control program should send IP configuration data to the sonar head and/or SIM to correct the issue.

```
struct R2DS                 // R2Sonic Device Status
{
  u32  PacketName;          // 'R2DS'
  u32  SerialNumber[3];     // up to 12 ASCII chars, unused chars are zero
  u32  ConfigID;            // from most recent R2DC packet
  u32  spare;
} pkt;
```

C structure of Device Status packet

Device status Ethernet packet example received from the sonar head

```
0000  00 e0 81 2e be 88 00 50  c2 90 40 58 08 00 45 00   .......P ..@ X..E.
0010  00 34 04 6c 00 00 32 11  6e 92 0a 00 00 56 0a 00   .4.l..2. n....V..
0020  01 66 ff 16 ff de 00 20  00 00 52 32 44 53 31 30   .f.....  ..R2DS10
0030  30 31 30 31 00 00 00 00  00 00 46 35 bd 01 00 00   0101.... ..;.~...
0040  00 00                                              ..
```

```
0000  00 e0 81 2e be 88 00 50  c2 90 40 49 08 00 45 00   .......P ..@ I..E.
0010  00 34 02 75 00 00 32 11  70 7c 0a 00 00 63 0a 00   .4.u..2. p|...c..
0020  01 66 ff 7a ff de 00 20  00 00 52 32 44 53 31 30   .f.z...  ..R2DS10
0030  30 30 34 34 00 00 00 00  00 00 46 35 bd 01 00 00   0044.... ..;.~...
0040  00 00                                              ..
```

Device status Ethernet packet example received from the SIM

15.14 Data Playback Using Bit-Twist

15.14.1 Introduction

Note, the topics covered in this document require knowledge of Ethernet communication.

To test a data collection system, you can either use the actual hardware (sonar head) or use data captured from the sonar head. Using Wireshark, uplink data from the sonar head can be captured, filtered, and saved. Bit Twist, a console application, allows you to playback data. R2Sonic can supply sample Ethernet captures of the sonar head uplink data. You may need to edit the destination MAC and IP addresses of the captured data with Bit-Twiste, a console application. Wireshark and Bit-Twist both require Winpcap which is included in the Wireshark installation.

In the examples, the following IP addresses are used:

Sonar head: 10.0.0.86
Data collection computer: 10.0.1.102

The following programs are required:

To capture, filter, and save Ethernet data:
Wireshark: http://www.wireshark.org/
To playback and edit captured Ethernet data:
Bit-Twist: http://bittwist.sourceforge.net/
Using a 32-bit version of Wireshark will allow you to use a packet decoder for the sonar data formats.

If you don't want or need to install Wireshark, get Winpcap at:
Winpcap: http://www.winpcap.org/

15.14.2 Capturing Data

To capture data from the sonar head, use Wireshark. Set the max ping rate of the sonar to 1 to 5 pings per second so you won't create huge capture files.

- Capture sonar data. For high data rate traffic, set the following Wireshark Capture Options. These options are found under the button (usually left most) "List the available capture interfaces…". These setting will remain for the session.
  Buffer size: 50 megabytes
  Uncheck "Update list of packets in real time"

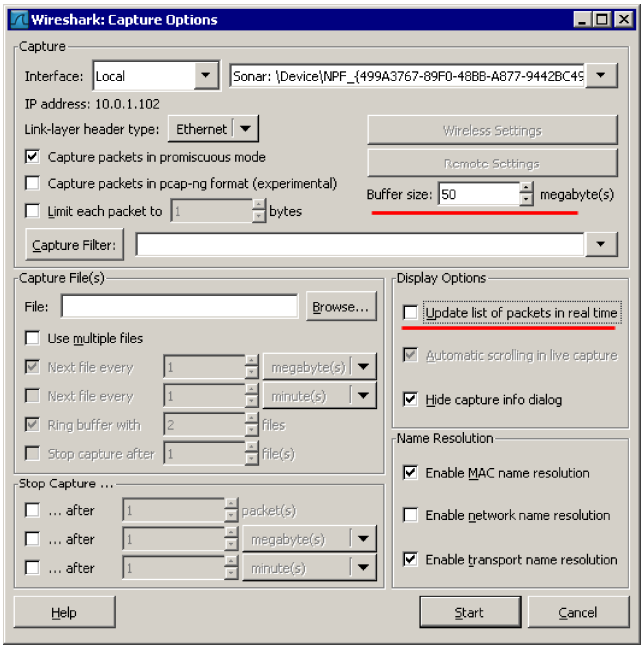

Figure 206: Wireshark Capture Options

This will reduce the processing load on Wireshark significantly.

•After capture, filter the data so only the desired sonar head data is displayed. A filter expression like

"not(icmp.type == 3 or ip.src == 10.0.1.102)"

can be used to filter data coming from the data acquisition computer.

•Save using Save As, data type as "Wireshark/tcpdump/…- libpcap (*.pcap,*.cap)" (Wireshark default). Select "Displayed" in Packet Range. You can select a data range in the Packet Range such that the data packets aren't truncated.

15.14.3 Editing Data

The MAC and IP addresses in the packets must match the data acquisition computer's MAC and IP addresses assigned to the network interface card (NIC). The data acquisition computer's MAC and IP addresses can be determined using ipconfig /all from the command line.

Editing the MAC and IP addresses must be done as separate operations using bittwiste.exe. The following examples show the syntax for editing the destination MAC and IP address in the .pcap files created by Wireshark.

Example to change destination MAC address using bittwiste.exe:

```
bittwiste -I in.pcap -O out.pcap -T eth -d 00:E0:12:7F:D2:1A
```

Example to change destination IP address using bittwiste.exe:

```
bittwiste -I in.pcap -O out.pcap -T ip -d 10.0.1.102
```

Where in.pcap is the input file and out.pcap is the output file.

15.14.4 Data Playback

To playback data, use bittwist.exe. You can playback data on the same computer that the data collection program resides on by simply connecting the sonar Ethernet port to an Ethernet switch. The Ethernet switch is only to placate the NIC. You can also send data from a remote computer to the data acquisition computer.

The Ethernet interface number needs to be determined. Choose the interface that is connected to the sonar system. To display Ethernet interfaces:

```
bittwist –d
```

To playback data:

```
bittwist -i 2 -l 0 out.pcap
```

This sends out.pcap to Ethernet interface 2 (-i 2) and loops continuously (-l 0). Use Ctrl-C to exit the program.

If you don't want to loop, use:

```
bittwist -i 2 out.pcap
```

Note:
Newer Ethernet playback utilities are better than *bittwist*. Ask support about a modified version of *Playcap.exe* (Open Source program modified for looping) and *Wireplay.exe* (R2Sonic program that can play back pcap files at full speed and you don't need to edit pcap files with *bitttwiste*). *Wireplay.exe* is the recommended playback utility. Wireplay is available from R2Sonic.