

Appendix VI Sonic Control Commands

13 R2Sonic Control Commands

13.1 Introduction

The Sonic 2024/2022 is configured and controlled using a set list of commands. Sonic Control works by sending these commands to the system. The control commands, provided here, can be used to control the Sonic 2024/2022. This describes the commands sent from the user interface to the sonar head and SIM.

Head firmware version: Head\$14Mar2011-20-44-49.bin and SIM firmware version: Simb\$24-sep-2010-16-11-31.bin utilise the commands in this document. Future versions of firmware will adhere to this format and may include additional commands.

Older versions of head and SIM firmware are **not** compatible with this format.

13.2 General Notes

1. 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. u32 means unsigned integer, 32 bits.
f32 means IEEE-754 32-bit floating point.
4. All packets are UDP/IP datagrams.
5. It's recommended that all commands be sent periodically, at a 1 to 0.5 Hz rate. This ensures that the sonar head and SIM always have the proper settings should a power interruption occur.

13.2.1 Port Number

Port = Baseport +2 where the baseport is the sonar head or SIM baseport

13.2.2 Type Definitions

```
typedef unsigned int u32;
typedef float f32;
```

13.2.3 Command Packet Format

Pseudo C format for commands

```
// *** BEGIN PACKET: COMMAND FORMAT 0 ***
u32 PacketName;           // 'CMD0'

// Command (for network efficiency, the packet can contain multiple commands,
// but ensure the IP datagram reaches the sonar unfragmented).

u32 CommandName;          // example 'RNG0' to set range
x32 CommandValue;         // a 4-byte value such as u32 or f32

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

Version	3.1	REV	r000
Date	01-04-2011		

13.3 R2Sonic Head Commands – Binary Format

Cmd	Format	Units	Values	Description
ABS0	f32	dB/km	0 to 200	absorption
AUT0	u32		Flag bits 0x00000001 = auto power on 0x00000100 = auto gain on 0x00010000 = auto range on	[7:0] auto power [15:8] auto gain [23:16] auto range [31:24] spare Auto power/gain work in tandem, thus both are enabled/disabled at same time.
BIE0	u32		0 = off 1 = on	bathy intensity enable
BMAX	f32	metres	0 to 999	max range filter head default = 999
BMIN	f32	metres	0 to 500	min range filter head default = 0
BOS0	u32		0 = Equiangle 1 = Equidistant	bottom sampling
DGA0	f32	metres	0 to 500	depth gate min
DGB0	f32	metres	0 to 500	depth gate max
DG00	u32		0 = off 1 = on 0x00000000 = gates off 0x00000001 = manual gates 0x0000ww10 = auto gates 0x0000ww11 = auto gate/slope where ww = gate width in $\pm$ percent of depth (5% to 90%)	depth gates control. Manual gates mode require DGA0, DGB0, DGS0 to be set. In auto gates mode, a peak percentage value for gate width must be supplied in bits [15:8] in this command.
DGS0	f32	radians	$-\pi/2$ to $+\pi/2$	depth gate slope
DHM0	u32		0 = single head 1 = master simultaneous 2 = master alternating 3 = slave simultaneous 4 = slave alternating	head sync mode (single and dual head modes)
FILT	u32		0 = off 1 = range 2 = depth 3 = range & depth	FILT is depreciated. Do not use (10 Mar 2011). Use DG00.
FRQ0	f32	Hz	200000 to 400000	frequency
GAN0	f32	dB * 2	1 to 45	gain ($\Delta 1 = 2\text{dB}$)
PRL0	f32	Hz	0.1 to 60	ping rate limit user-value
PR00	u32		0 = projector forward 1 = projector aft	projector orientation
PROJ	u32		0 = none 1 = narrow (1°) 2 = wide (20°)	projector type selector
PRU0	u32		0 = off 1 = on	ping rate limit user-enable
PRZ0	f32	metres	-1.0 to +1.0	projector mounting Z offset default = 0.119
RET0	f32	radians	-45° to +45°	receiver tilt

Cmd	Format	Units	Values	Description
RNGØ	f32	metres	2 to 500	range
ROSØ	u32		0 = off 1 = on	roll stabilization enable
SERØ	f32	radians	-70° to +70°	sector rotate wedge edges must not go beyond ±80°
SEWØ	f32	radians	10° to 160°	sector width
SNIP	u32		0 = off 1 = on	snippets enable
SPRØ	f32		0 to 60 typically 20	spreading loss
SVLØ	f32	m/s	1250 to 1600	sound velocity user-value
SVUØ	u32		0 = use SVP 1 = user value	sound velocity user-enable
TWIX	u32		0 = flat bottom 1 = vertical features	bottom type
TXLØ	f32	seconds	0 to 1000µs	pulse length
TXPØ	f32	dB/1µPa	0, 191 to 221	power

13.4 R2Sonic SIM Commands – Binary Format

Cmd	Format	Units	Values	Description
BDGØ	u32	bps	standard baud rates 300 to 115200	GPS baud
BDMØ	u32	bps	standard baud rates 300 to 115200	motion baud
BDSØ	u32	bps	standard baud rates 300 to 115200	SVP baud
DBGØ	u32		7 or 8	GPS data bits
DBMØ	u32		7 or 8	motion data bits
DBSØ	u32		7 or 8	SVP data bits
DRGØ	u32		0 = auto	GPS driver
DRMØ	u32		0 = auto	motion driver
DRSØ	u32		0 = auto	SVP driver
ENGØ	u32		0 = off 1 = on	GPS serial port enable
ENMØ	u32		0 = off 1 = on	motion serial port enable
ENSØ	u32		0 = off 1 = on	SVP serial port enable
PAGØ	u32		0 = none 1 = odd 2 = even	GPS parity
PAMØ	u32		0 = none 1 = odd 2 = even	motion parity

Cmd	Format	Units	Values	Description
PAS0	u32		0 = none 1 = odd 2 = even	SVP parity
POG0	u32		0 = rising 1 = falling	PPS edge
SP00	u32		0 = head power off 1 = head power on	sonar head power

Appendix VII: R2Sonic Data Format

14 R2Sonic Uplink Data Formats

14.1 . Introduction

This describes the data formats sent from the sonar head to the data collection software. The formats are given in pseudo C.

Head firmware version 12-Apr-2010 utilise the data formats in this document. Head firmware version 25-Mar-2010-11-14-27 uses the bathymetry packet format described 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 are different than the format described in this document.

14.2 . General Notes

1. Each bathy or snippet 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.

14.3 . Port Numbers

Bathymetry data port = Baseport + 0

Snippets data port = Baseport + 6

Device status port = Baseport + 2

Where the baseport is the data collection software baseport.

14.4 . Type Definitions

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

14.5 . Bathymetry Packet Format

```
// *** BEGIN PACKET: BATHY 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]
u32  H0_TxMiscInfo;        // to be determined
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;            // [ $\Delta H0\_RxGain = 1$ ; 2dB gain change]
f32  H0_RxSpreading;       // [dB (times log range in meters)]
f32  H0_RxAbsorption;      // [dB per kilometer]
f32  H0_RxMountTilt;       // [radians]
u32  H0_RxMiscInfo;        // to be determined
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
```

Version	3.1	REV	r001
Date	01-04-2011		

```
// 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[6];           // several to-be-determined values such as attitude info to assist the GUI display

// 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;         // several to-be-determined values such as attitude info to assist the GUI display
f32 A2_MoreInfo[6];           // several to-be-determined values such as attitude info to assist the GUI display
u16 A2_AngleStep[H0_Points];  // [radians] angle[n] = (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;         // [micropascals] intensity[n] = I1_Intensity[n] * 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;         // [seconds two-way]
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
```

Version	3.1	REV	r000
Date	01-04-2011		



```
// 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 ***
```

Version	3.1	Rev	r000
Date	01-04-2011		

Part No. 96000001

14.6 Snippet Format

```
// *** BEGIN PACKET: SNIPPET 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]
u32  H0_TxMiscInfo;        // to be determined
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;            // [ $\Delta H0$  RxGain = 1; 2dB gain change]
f32  H0_RxSpreading;       // [dB (times log range in meters)]
f32  H0_RxAbsorption;      // [dB per kilometer]
f32  H0_RxMountTilt;       // [radians]
u32  H0_RxMiscInfo;        // to be determined
u16  H0_reserved;          // reserved for future use
u16  H0_Snippets;          // number of snippets
f32  H0_MoreInfo[6];       // several to-be-determined values such as attitude info to assist the GUI display
```

Version	3.1	REV	r000
Date	01-04-2011		



```
// 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]; // [Magnitude] = 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 FORMAT 0 ***
```

Version	3.1	Rev	r000
Date	01-04-2011		