Sonar Data Stream Protocol Manual V1.1

Marine Sonic Technology, Ltd.
White Marsh, VA

This product was designed and developed by a team of engineers at Marine Sonic Technology, Ltd.

Marine Sonic Technology, Ltd.
5508 George Washington Memorial Highway
P.O. Box 730, White Marsh, VA 23183-0730
(804) 693-9602
(800) 447-4804

Technical Support

For technical support call (804) 693-9602 or visit our web site at http://www.marinesonic.com.

Copyright

Trademarks

Table of Contents

Section 1 Introduction 2

Section 2 The Sonar Data Stream In Depth 4

1 Variable Types 4

2 General Packet Structure 4

3 Data Packet Types 6

Sync Data Packet 6

Synchronizing the Data Stream 7

Calculating Real Time 7

Sonar Data Packet Version 1 8

Sonar Data Packet Version 2 10

Navigation Data Packet 13

Orientation Data Packet 14

Ship, Sonar, and Sensor Positions 14

Fathometer Data Packet 15

Magnetometer Data Packet 16

Mark Data Packet 16

NMEA Data Packet 17

Custom Data Packet 17

Types of Data Sent By Sea Scan PC Systems 17

Types of Data Sent By Sea Scan HDS Systems 17

Types of Data Sent By Sea Scan ARC Explorer Systems 18

4 Reading and Writing the Sonar Data Stream 18

Data Structure Packing 18

Files 18

Network Packets 19

Serial Packets 19

Section 3 Appendix 21

1 SDS.H 21

2 Contact Information 33

Section 1

- **Introduction**

1 Introduction

This manual describes the Sonar Data Stream data transport protocol. Sonar Data Stream files contain Sonar Data Stream formatted 4 packets. Since these packets were designed to be able to be transmitted over multiple mediums reading the SDS stream from a file 18 is similar to reading it from a network connection 19 or a serial data stream 19.

Sea Scan HDS utilizes the Sonar Data Stream exclusively for data transport and storage. This allows for a unified easy to use data interface capable of meeting all of MSTL's sonar data transport needs. In addition to storing sonar data, the Sonar Data stream stores navigation, orientation, NMEA strings, and target marking data. This allows the data stream to store all of the data collected by Sonar Data collection applications eliminating the need for many external files. The Sonar Data Stream's structure allows it to store only the data that is needed eliminating the extra empty fields that waste space and cause confusion that other sonar data exchange formats cause. The sonar data also features the ability to store custom data packets which can coexist with the standard data packets. This allows 3rd party applications to utilize the Sonar Data Stream format for their own applications.

Section 2

- **The Sonar Data Stream In Depth**

2 The Sonar Data Stream In Depth

The Sonar Data Stream is a data format that allows the inclusion of sampled sonar data of various types, navigational and positioning data, fathometer (depth & altitude) data, magnetometer data, as well as markers or notable events all in the same data stream. The most important concept to understand in the Sonar Data Stream is the time stamping system. Every piece of data is stamped with a millisecond accurate (theoretically) timestamp that is reference to Greenwich Mean Time. This allows the data to be processed by other applications in accordance to the time at which the data was collected.

The Sonar Data Stream was engineered to be serialized and streamed into/out of a file, over network connections of various types, and over asynchronous and synchronous serial links. It includes a synchronization scheme that allows for a stream that is interrupted to resume receiving again at a known good location in the data.

This section provides a fairly good description of the Sonar Data Stream and how it should be used. You may find looking at the C/C++ header file more informative or easier to digest. Please refer to the **SDS.H** 21 part of the Appendix for the header file. If you would like more information, need help with some of the concepts of the Sonar Data Stream or would like to propose an addition or modification please **contact us** 33.

2.1 Variable Types

All of the data contained in the sonar data stream is stored as "Little Endian" byte order. The variable sizes that are used in the **header file** 21 as well as in the following section(s) are listed below.

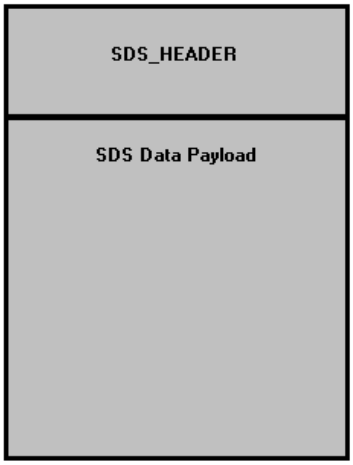
Type	Size in Bytes	Signed/Unsigned
BYTE	1	unsigned
WORD	2	unsigned
DWORD	4	unsigned
int	4	signed
short int	2	signed
float	4	NA
double	8	NA

2.2 General Packet Structure

Each Sonar Data Stream packet contains a header and a data payload (optional). The SDS_HEADER is 16 bytes in size and is formatted like this:

Element	Variable Type	Description
dwSize	DWORD	The size of the entire data packet, not including this header.
dwTimeStamp	DWORD	The number of milliseconds since Midnight GMT.
dwTag	DWORD	This describes the contents of the data packet and how the header will be used.
byMisc[3]	BYTE	Used for miscellaneous storage.
byChecksum	BYTE	This is a simple 8 bit checksum calculated by adding all of the bytes in the packet header (for a total of 15 bytes) into an 8 bit number. This checksum excludes the byChecksum variable (the last byte).

Sonar Data Stream Packet



In addition to the header a packet may have a data payload. The tag is used to describe the packet type. In the tag's description there will be information on how and what the data is. A packet's data always follow the header.

Each packet header must have the size, tag, and time stamp filled out. The miscellaneous storage should be zeroed out for every packet type except for the Sync packet. The Sync packet should have the miscellaneous storage filled with a value of 0xAA.

There are currently 7 possible values for the tag. These values are listed in the next section, Data Packet Types 6. Additional data packets may be defined (if inserting your own data into the sonar data stream). Packets that contain an invalid tag should be ignored. This can be accomplished by simply skipping over the data payload using the dwSize variable in the packet's header.

Built into the Sonar Data stream is a couple of mechanisms to insure the stream has valid data. The first method, which is mentioned above, is to read only the tags that your software understands. Another mechanism is the 8 bit checksum if the packet header that is at the end of the header. This will give a program a good indication of the validity of a packet received. If the checksum should come out to be bad, **re-synchronizing the data stream** 7 will be necessary.

In the newest version of the data stream we have added an 8 bit data checksum. This checksum can be applied to every packet except for the Sync data packet. The 8 bit data checksum is stored in the byMisc[2] (3rd misc. byte). If the byte is 0 then the checksum is ignored and the data packet is assumed to be good. if the byte is > 0 then the this represents an 8 bit checksum of the data payload section of the Sonar Data Stream packet. To calculate this checksum simply add all of the data bytes in the data payload to an 8-bit unsigned variable. Compare this result to the transmitted checksum value in the byMisc[2] variable. If the 2 are not the same then the data payload is assumed to be faulty and should not be used.

Skipping back and forth in the data stream can be accomplished in chunks. By using the dwSize variable which is at the beginning of every sonar data stream packet you can rewind and skip ahead to different sections of the data stream. Please note that skipping ahead is only an option if the data stream is being read from a non-actively written to file on disk.

2.3 Data Packet Types

The are 7 basic packets types defined right now in the sonar data stream. Each of these packets has an associated tag that is 32 bits in size. List below are the 7 types and their tags.

Type	Tag	Description
Sonar Version 1 8	0x534E5200	This is a sonar data packet. It can store the data types of many different sonar systems as well as many different data formats.
Sonar Version 2 10	**0x32524E52**	This is a sonar data packet version 2. It is a newer better version of the Sonar Data Packet Version 1.
Sync 6	0x53594E43	This tag represents a sync data packet. This packet is sent at regular intervals and is used to synchronize the data stream.
Navigation 13	0x4E415600	This data packet allows the storage of navigation data.
Orientation 14	0x4F524E54	This data packet stores orientation of an data source from the origin of the navigation data.
Fathometer 15	0x46415400	This data packet stores fathometer information such as depth & altitude of an object.
Magnetometer 16	0x4D414700	This data packet stores magnetometer readings.
Mark 16	0x4D41524B	This data packet allows marks to be placed in the sonar data. The marks are time as well as geographically referenced.
NMEA 17	0x4E4D4541	The NMEA data packet stores strings containing commands and data whose format is based on the NMEA 0183 standard.
Custom 17	**0x00######**	The custom data packet is available for manufacturer specific data and should be ignored if not recognized. The custom tag allows for 3 8 bit numbers that can be used to form the custom tag.

2.3.1 Sync Data Packet

The Sync data packet allows the data stream to be synchronized. It can be used to find a good starting point in corrupted data or it can be used to start a frame. Skipping to the next Sync packet will the advance the frame X amount of time, where X is the Sync packet interval. The SYNC packets should be sent at regular intervals which are specified in the packet's data payload.

Every stream should start with SYNC packet. This is to prevent data from unnecessarily being skipped

over by the synchronization process. To learn more about how to synchronize the data stream please refer to **Synchronizing the Data Stream** 7 later in this section.

Please note that the Sync data packet is the only packet that uses the miscellaneous bytes in the packet header. The bytes should be assigned the 0xAA value. This value insures that the data stream can be re-synchronized should it become corrupted. For more information see the Synchronizing the Data Stream section. This will yield 3 consecutive 0xAA values.

A Sync data payload is defined in the table below:

Element	Variable Type	Description
dwReference	DWORD	This is the number of seconds elapsed since midnight (00:00:00), January 1, 1970
wInterval	WORD	This is the sync packet repetition interval in milliseconds.

2.3.1.1 Synchronizing the Data Stream

There are two specific times at which you will want to synchronize the sonar data stream. The first is when an application first starts reading data from its network connection. The second is when a sonar data stream becomes unsynchronized (a header checksum fails).

In order to synchronize the data stream, start by parsing the stream for the serial synchronization bit stream (0x53594E43AAAAAA). The bit stream consists of the SYNC tag and the serial synchronization DWORD. This unique bit stream identifies a SYNC packet. After the synchronization bit stream is confirmed valid the 4 bytes before make the dwSize variable. You should now rewind to the start of the packet header and confirm that the header is a valid one. You can then read the reference time stamp to get the exact date and time of this packet as well as when to expect the next SYNC packet.

2.3.1.2 Calculating Real Time

The Sync data packet is key to generating a real time timestamp for the rest of the data. Without the reference time stamp the data stream would just be able to calculate elapsed time. The calculation for the actual real time is a simple one is based on a frame of data. A frame of data is 1 second worth of time in the SDS stream. Each frame consists of a Sync data packet and SDS data until a new Sync data packet is received. The Reference Timestamp and Relative Timestamp of the Sync data packet are used as the reference time to which all of the other SDS data packets in the frame are referenced to. The following are the calculations for the SDS time:

Offset Milliseconds = (SDS Data Packet Relative Time) - (SYNC Packet Relative Time)
Real Time In Seconds = (SYNC Packet Reference Time) + ((Offset Milliseconds) / 1000.0)

This yields a real time in decimal seconds with resolution down to 0.001 second (1 millisecond). This time represents the number of decimal seconds elapsed since midnight January 1st, 1970.

The data stream is generally in chronological order. However please note that due to data coming potentially different sources the ***Offset Milliseconds*** may be a negative number. This is not an error.

Using the above calculation the time will still come out correctly. This is caused by another data source that is inserting data into the data stream.

2.3.2 Sonar Data Packet Version 1

The Sonar data packet allows the storage of multiple types of sonar data. The sonar data starts with an sonar data channel structure followed by the actual sonar data itself. Each individual channel has its own Sonar data packet. Please note that the Sonar Data Packet is version 1 and is not recommended for new sonar designs. Please see the Sonar Data 2 packet if you are starting a new sonar design.

The sonar data channel structure is as follows:

Element	Variable Type	Description
wSonarType	WORD	This is the type of sonar/ transducer (Port, Starboard, Forward Looking, Scanning, etc.)
fFreqHz	float	This is the acoustic frequency in Hertz of this channel.
fSpeedofSound	float	This is the speed of sound in water. It is in meters per second.
fRangeDelayMs	float	This is the delay from the start of the transmit pulse to the first sample in milliseconds.
fRange	float	This is the range in milliseconds that this set of data represents.
bySampleSizeBytes	BYTE	This is the size in byes of each sonar sample. This will depend heavily on the type of data being stored.
wDataType	WORD	This is the type of data stored in this data set. Please see the possible data types that are listed below.
dwSamples	DWORD	This is the number of samples in this channel.

The Sonar Type can be one of several types. The different types of data is listed below:

Sonar Type	wSonarType Value	Description
Port Side Scan	0x0001	Port facing side scan sonar.
Starboard Side Scan	0x0002	Starboard facing side scan sonar.

Forward	0x0003	Forward looking sonar.
Multibeam	0x0004	Multi-Beam sonar.
Scanning	0x0005	Scanning sonar.
Sub-Bottom	0x0006	Sub-Bottom Profiling sonar.
User Defined	0x0100	User defined sonar. If you don't know what it is, don't parse it.

For sonar types such as scanning sonar, the **orientation data packet** 14 describes the direction in which the scanning sonar is pointing and it's orientation with reference to the ship or other object.

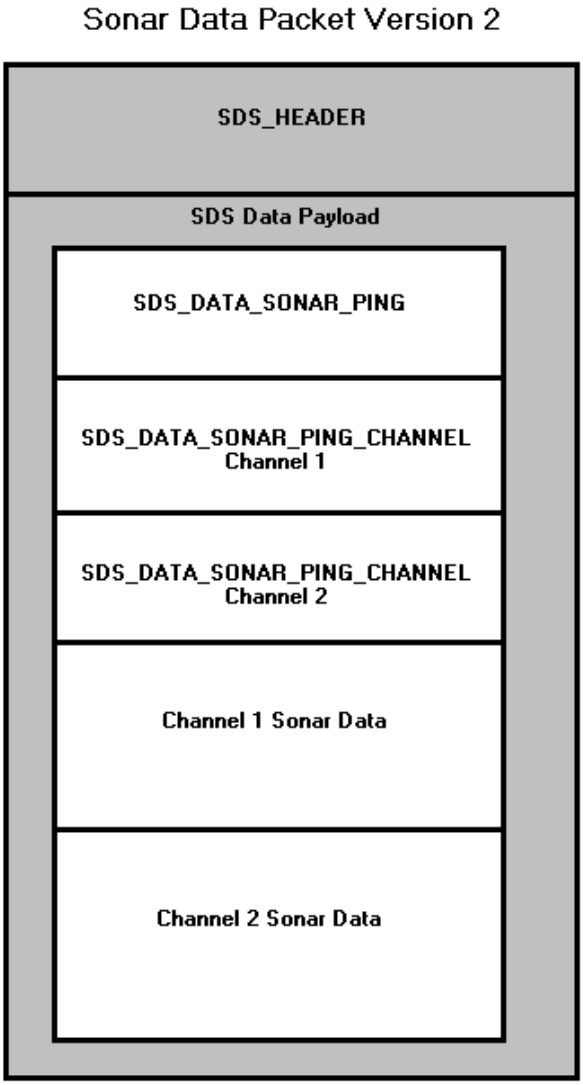
The data type also has many different flavors. The data type affects the sample size. You can use the bySampleSizeBytes along with the dwSamples to calculate the exact number of bytes that follows the sonar data channel structure. The different data types are listed in the table below:

Data Type	wDataType Value	bySampleSizeBytes Value	Comments
Unsigned 8 Bit	0x0001	1	
Signed 8 Bit	0x0002	1	
Unsigned 12 Bit	0x0003	2	The upper 4 bits should be 0.
Signed 12 Bit	0x0004	2	The upper 4 bits should be 0 (except for the sign)
Unsigned 16 Bit	0x0005	2	
Signed 16 Bit	0x0006	2	
Floating Point	0x0007	4	
Complex Integer	0x0008	4	A complex integer is stored as two consecutive signed 16 bit integers, the real element first, followed by the imaginary element.
Complex Float	0x0009	8	A complex float is stored as two consecutive floating point numbers, the real element first, followed by the imaginary element.

2.3.3 Sonar Data Packet Version 2

The Sonar Data 2 packet, like the Sonar Data packet allows the storage of multiple types of sonar data. The Sonar Data 2 packet besides being more efficient at storing the sonar data also allows more than 1 channel of sonar data to be stored within the packet. The sonar data starts with a SDS_DATA_SONAR_PING structure. The SDS_DATA_SONAR_PING structure describes some general sonar parameters and contains the count of SDS_DATA_SONAR_PING_CHANNEL structures that follow the SDS_DATA_SONAR_PING structure. The SDS_DATA_SONAR_PING_CHANNEL structures immediately follow the ping structure. The is 1 SDS_DATA_SONAR_PING structure for each channel and are in order starting at 1 to the byChannelCount. The sonar data itself follows the list of channel structures.

A block diagram showing the general arrangement of data structures in the Sonar Data 2 packet is pictured to the right. Each data structure that the Sonar Data Packet Version 2 uses is described below.



The SDS_DATA_SONAR_PING structure is as follows:

Element	Variable Type	Description
byChannelCount	BYTE	This variable describes the number of sonar data channels present in this sonar data packet.
dwPingNumber	DWORD	This number is a counter that increments by 1 for each sonar data packet transmitted.
fSpeedofSound	float	This is the speed of sound in water. It is in meters per second.
byReserved[7]	BYTE array	This array is reserved for later use. It reserves space that may be used to create new structure elements in

		the future. These should be set to 0.

Each SDS_DATA_SONAR_PING_CHANNEL structure is described by the sonar ping channel structure described below:

Element	Variable Type	Description
wSonarType	WORD	This is the type of sonar/transducer (Port, Starboard, Forward Looking, Scanning, etc.) See below for a list of the possible sonar types and their descriptions.
wSonarID	WORD	This variable serves as a unique identifier to distinguish this transducer from other transducers. For example it can store a transducer serial number or position. If not being used it should be set to 0.
fFreqHz	float	The acoustic center frequency in Hertz for this transducer.
fRangeMs	float	The range in milliseconds that this channel represents.
fRangeDelayMs	float	This is the delay from the start of the transmit pulse to the first sample in milliseconds.
wDataFlags	WORD	The data flags provide additional information about the sonar data. See below for a list of flags.
wDataType	WORD	This is the type of data stored in this data set. Please see below for a list of the different data types and their descriptions.
wSamples	WORD	This is the number of samples in this channel.

The Sonar Type can be one of several types. The different types of data is listed below:

Sonar Type	wSonarType Value	Description
Port Side Scan	0x0001	1st (or only) Port facing side scan sonar channel.
Starboard Side Scan	0x0002	1st (or only) Starboard facing side scan sonar.
Port Side Scan 2	0x0010	2nd Port facing side scan sonar channel.
Starboard Side Scan 2		2nd Starboard facing side scan sonar channel.
Forward	0x0003	Forward looking sonar.
Multibeam	0x0004	Multi-Beam sonar.
Scanning	0x0005	Scanning sonar.
Sub-Bottom	0x0006	Sub-Bottom Profiling sonar.
User Defined	0x0100	User defined sonar. If you don't know what it is, don't parse it.

For sonar types such as scanning sonar, the **orientation data packet** 14 describes the direction in which the scanning sonar is pointing and it's orientation with reference to the ship or other object. Currently only the Port Side Scan and Starboard Side Scan sonar types are being used by Sea Scan PC and Sea Scan HDS side scan sonar systems.

In addition to the different types of sonar data that can be stored in the Sonar Data Packet, there are different Data Types that can be stored as well. This allows for the most flexibility when implementing a Sonar Data Stream transmitter. The Data The Data Type describes the format of each sonar data sample. This format determines the number of bits and the type of binary number that is represented by the sample. In order to calculate the number of bytes stored in a channel's data, use the sample size in bytes and multiply times the number of samples in the channel structure. If there is a number after the decimal point such as when using a 12 bit packed Data Type, simply always round up (the left over nibble can be then thrown out when reading the data). Currently the Unsigned 8 Bit and Unsigned 12 Bit Packed are the only data types being used in Sea Scan PC and Sea Scan HDS Side Scan Sonar systems. The different Data Types are listed in the table below:

Data Type	wDataType Value	Sample Size in Bytes	Comments
Unsigned 8 Bit	0x0001	1	
Signed 8 Bit	0x0002	1	
Unsigned 12 Bit Packed	0x0003	1.5	This data is stored 3 nibbles wide. The high nibble of the 1st sample should be the low nibble of the next byte.
Signed 12 Bit Packed	0x0004	1.5	This data is stored 3 nibbles wide. The high nibble of the 1st sample should be the low nibble of the next byte.
Unsigned 12 Bit Unpacked	0x000A	2	This data is stored 4 nibbles wide. The upper nibble in the most significant byte is 0.
Signed 12 Bit Unpacked	0x000B	2	This data is stored 4 nibbles wide. The upper nibble in the most significant byte is 0.
Unsigned 16 Bit	0x0005	2	
Signed 16 Bit	0x0006	2	
Floating Point	0x0007	4	
Complex Integer	0x0008	4	A complex integer is stored as two consecutive signed 16 bit integers, the real element first, followed by the imaginary element.

Complex Float	0x0009	8	A complex float is stored as two consecutive floating point numbers, the real element first, followed by the imaginary element.
Unsigned 32 Bit	0x000C	4	
Signed 32 Bit	0x000D	4	

The Sonar Data Packet Version 2 adds an additional description variable that is used to add more meaning to the type of data stored in the channel's data. This is the wDataFlags variable. The Data Flags add a useful description of the sonar sample that indicates what kind of data is stored and is useful when processing and displaying the data. List below are the current Data Flags that are described in the SDS standard:

Data Flag	wDataFlag Value	Description
None	0x0000	Nothing special about this data.
Raw Data	0x0001	This data has not been processed.
Log Compressed	0x0002	This data has been logarithmically compressed and is not linear.
TVG	0x0004	This data has had TVG applied.
Even Data	0x0008	This data is interlaced and represents the even samples for this channel I.E. : 0, 2, 4, 6, 8 ... The other half of this data is contained in another Sonar Data Packet with the Odd Data Data Flag set.
Odd Data	0x0010	This data is interlaced and represents the odd samples for this channel I.E.: 1, 3, 5, 7, 9 ... The other half of this data is contained in another Sonar Data Packet with the Even Data Flag set.

2.3.4 Navigation Data Packet

The navigation data packet allows the storage of navigation data. This navigation data is strictly geographically defined using standard latitude/longitude coordinates. The navigation data payload structure is listed below.

Element	Variable Type	Description
wSource	WORD	Origin of this navigation data.
dLatitude	double	The latitude encoded as decimal degrees.
dLongitude	double	The longitude encoded as decimal degrees.

fCOG	float	The course over ground in degrees. (Stored as TRUE not magnetic)
fHeading	float	The heading in degrees (Stored as TRUE not magnetic)
fSOG	float	The speed over ground in meters per second.

Please see the Ship, Sonar and Sensor Positions 14 section for more information on values to place in and how to use the source fields.

2.3.5 Orientation Data Packet

The orientation data packet describes the orientation of the sonar in relation to the navigation data. The source of the navigation data is described in the data payload which is described below.

Element	Variable Type	Description
wSource	WORD	This is the origin of the navigation data which this packet adds to.
fX	float	X Offset in meters form the origin.
fY	float	Y Offset in meters form the origin.
fZ	float	Z Offset in meters from the origin.
fRoll	float	Roll in degrees.
fPitch	float	Pitch in degrees
fYaw	float	Yaw in degrees.
fHeave	float	Heave in centimeters.

Please see the Ship, Sonar and Sensor Positions 14 section for more information on values to place in and how to use the source fields.

2.3.5.1 Ship, Sonar, and Sensor Positions

Each data packet with the exception of the Sonar and Sync data packets have a source field. This is used to describe the origin of the data packet. These packets are typically created from devices that are positioned away from the origin of a data collection system and may be rotated. In order to describe the orientation of these sensors as well as describe the orientation of the sonar Orientation Data Packet is used. The Source field in each data packet simply tells us where our data came from and the Orientation data packet describes how the Source is oriented in reference to a an origin about which this system is constructed. The Fathometer and Navigation data packets describe the sources locations in the real world.

The source is stored in the wSource variable. A table of these sources and their identifier value can be found below.

Navigation Source	wSource Value	Description
Ship	0x0001	The navigation data came from the survey ship.
Sonar	0x0002	The navigation data came from the sonar itself (usually a towfish or AUV).
Sensor 1	0x0003	The navigation data came from sensor number 1.
Sensor 2	0x0004	The navigation data came from sensor number 2.
Sensor 3	0x0005	The navigation data came from sensor number 3.
Sensor 4	0x0006	The navigation data came from sensor number 4.

There are several applications that combine the use of the Source and Orientation data packets. One example of this is the use of a Rotating Sonar Head. In such a system the Orientation data packet would list its source as Sonar and would contain Yaw data. This indicates which direction the sonar head was pointed in. In this application there would be 1 Orientation data packet per Sonar data packet. Another example is in describing the layback of a towed sonar system in reference to the towing vessel. This system would include an Orientation data packet that lists the Sonar as the Source. The navigation data would then be recorded with the Ship as the source. This combination of data packets then describes the offset of the towed sonar system from the ship.

2.3.6 Fathometer Data Packet

The fathometer data packet basically stores depth and altitude information for a particular origin. This data can be used along with the navigation data to give the user an accurate geographical location. The fathometer data payload is described below.

Element	Variable Type	Description
wSource	WORD	This is the origin of the fathometer data.
fDepth	float	This is the depth in meters.
fAltitude	float	This is the altitude in meters.

Please see the Ship, Sonar and Sensor Positions 14 section for more information on values to place in and how to use the source fields.

2.3.7 Magnetometer Data Packet

The magnetometer data packet stores readings from a magnetometer sensor. It contains a source variable to indicate where the magnetometer is located. Please see the sources listed in **Navigation Data Packet** 13 section for possible values for wSource. Below is the structure of the magnetometer data payload.

Element	Variable Type	Description
wSource	WORD	This variable stores the source location of the magnetometer data.
fReading1	float	This is the reading number 1 from the magnetometer in Gammas.
fReading2	float	This is the reading number 2 from the magnetometer.

Please see the Ship, Sonar and Sensor Positions 14 section for more information on values to place in and how to use the source fields.

2.3.8 Mark Data Packet

The marker data packet is used to store information particular to a geographical point at a particular time. This can be used to place annotations, markers, and way points in the sonar data stream. The marker data packet contains a description string as well as a location in Latitude/Longitude plus offset information. The marker data packet consists of the marker data payload structure and is followed by a NULL terminated string. The length of the string is contained within the marker data payload. A description of the payload can be found below:

Element	Variable Type	Description
dLat	double	Origin Latitude of the mark.
dLon	double	Origin Longitude of the mark.
fXOffset	float	Offset in meters in the X direction from the origin.
fYOffset	float	Offset in meters in the Y direction from the origin.
fZOffset	float	Offset in meters in the Z direction from the origin.
wLength	WORD	Length of the string in the characters including the NULL terminator.

2.3.9 NMEA Data Packet

The NMEA data packet stores strings containing commands and data whose format is based on the NMEA 0183 standard. The NMEA data packet consists of the NMEA data payload structure and is followed by a NULL terminated string. The NMEA data payload contains the length of the string and contains a source variable to indicate the device from which the string originated. Please see the sources listed in **Navigation Data Packet** 13 section for possible values for wSource. Below is the structure of the NMEA data payload.

Element	Variable Type	Description
wSource	WORD	This variable stores the source location of the magnetometer data.
wLength	WORD	Length of the string in the characters including the NULL terminator.

Please see the Ship, Sonar and Sensor Positions 14 section for more information on values to place in and how to use the source fields.

2.3.10 Custom Data Packet

Custom data packets can be fashioned by utilizing the maufacturer specific Sonar Data Stream Tag range using 0x00####### where the # is a free nibble for identification. Simply define a custom Tag to use in the SDS Header and then fill in your data payload with the custom data. Be sure to fill the rest of the SDS Header fields in correctly. When this data stream is processed by another Sonar Data Stream reader, the reader will skip over unrecognized SDS Tags. This ensures compatibility across a range of products.

2.3.11 Types of Data Sent By Sea Scan PC Systems

Sea Scan PC does not send all of the types of data that the sonar data stream supports. Below is a list of data packets that Sea Scan PC does send:

- Sync Data Packet
- Sonar Data Packet Version 1 - Port Side Scan Sonar - 8 Bit Unsigned
- Sonar Data Packet Version 1 - Starboard Side Scan Sonar - 8 Bit Unsigned
- Navigation Data Packet - Ship Source
- Navigation Data Packet - Sonar Source
- Fathometer Data Packet - Sonar Source

2.3.12 Types of Data Sent By Sea Scan HDS Systems

The Sea Scan HDS system utilizes most of the Sonar Data Stream packet types. To facilitate development of software that read Sea Scan HDS Sonar Data Streams (so you do not have to implement every single option) we have compiled a list of data types that Sea Scan HDS systems currently utilize. This list can be found below:

- Sync Data Packet
- Sonar Data Packet Version 2 - 12 Bit Packed - Port and Starboard Side Scan Sonar

- Navigation Data Packet - Ship and Sonar sources.
- Fathometer Data Packet - Ship and Sonar sources.
- Orientation Data Packet - currently used for Roll and Pitch of the sonar - Sonar Sources
- NMEA Data Packet - Ship, Sonar, and Sensor Sources

2.3.13 Types of Data Sent By Sea Scan ARC Explorer Systems

The Sea Scan ARC Explorer system utilizes most of the Sonar Data Stream packet types. To facilitate development of software that read Sea Scan ARC Explorer Sonar Data Streams (so you do not have to implement every single option) we have compiled a list of data types that Sea Scan ARC Explorer systems currently utilize. This list can be found below:

- Sync Data Packet
- Sonar Data Packet Version 2 - Unsigned 32 Bit - Port and Starboard Side Scan Sonar
- Navigation Data Packet - Ship and Sonar sources.
- Fathometer Data Packet - Ship and Sonar sources.
- Orientation Data Packet - currently used for Roll and Pitch of the sonar - Sonar Sources
- NMEA Data Packet - Ship, Sonar, and Sensor Sources
- Custom ARC information packet (not listed in this document - see the Custom Data Packet 17 section).

2.4 Reading and Writing the Sonar Data Stream

The reading and writing of the Sonar Data Stream is a simple operation. Since was designed to be transported over a variety of digital data carrying mediums. These data mediums can generally be put into 3 categories: Files, Network Packets, and Serial Packets. Although the data is written to and read from these mediums in a similar fashion there are some general rules to follow in order to make it easier to do. Operations utilizing each of these mediums is described in the following sections.\

2.4.1 Data Structure Packing

One thing of importance to note is that the Sonar Data Stream structures are packed to 1 byte boundaries (I.E. no byte packing). Depending on your development platform you may have to configure the compiler to do this manually. Most 32 bit compilers default to packing data structures to 4 byte boundaries. For example if a structure is 21 bytes in size then the compiler will add extra bytes to it to make it 24 bytes in size. While this is great for improving memory access performance it does not work well when reading/writing files, network data, or serial data.

2.4.2 Files

Reading and writing files containing the Sonar Data Stream is straight forward and simple. The file must be opened in binary mode and is written to sequentially. In other words as a data packet comes in it is written to the file. The only rule that should be followed is a file should be started with a synchronization packet. This ensures that the data stream is synchronized immediately at the beginning of the file. Failure to do so will result in up to 1 second worth of lost data for a standard Sonar Data Stream reader. Reading from a file is accomplished by reading the file byte for byte until the data stream is synchronized at which point a header data structure can be read followed by a data payload read. This continues until the file is done or an error is encountered at which point the data stream should be re-synchronized. For more information on synchronizing the Sonar Data Stream please see the Synchronizing the Data Stream 7 section earlier in this manual.

2.4.3 Network Packets

Reading and writing the Sonar Data stream from a network connection is straight forward and simple. Writing the data stream to network packets (UDP or TCP) is accomplished by writing 1 SDS packet per network packet. This can easily be accomplished for most of the Sonar Data Stream packets except for the Sonar Data packet. These packets can easily exceed the Maximum Transmission Unit size (which is typically 1500 bytes for most network interfaces). Sonar Data packets should be start in its own network packet and should be split among sequential network packets. Reading data from a network packet should be accomplished by reading the network data until a synchronization packet is received. Once the data stream is synchronized the network packets then can be assembled into a Sonar Data Stream much in the same way it is read from files. When an error occurs the data stream should be re-synchronized. For more information on synchronizing the Sonar Data Stream please see the Synchronizing the Data Stream 7 section earlier in this manual. Since the Sonar Data Stream packets should be packed 1 per packet it is not entirely necessary to re-synchronize (it is a good idea). If performance requirements are high then the data packet with the error in it should be thrown away and the data steam should be resumed on the next data packet.

2.4.4 Serial Packets

Reading and writing the Sonar Data stream from a serial port is a bit more involved and can vary from situation to situation. Marine Sonic Technology recommends the use of a lower layer packet based transport mechanism such as SLIP or PPP to aid in the efficient transmission of Sonar Data Stream packets. After it is encapsulated in a SLIP or PPP serial stream the Sonar Data Stream can then be written and read in a similar manner to the way it is handled when working with Network Packets.

Section 3

- **Appendix**

3 Appendix

3.1 SDS.H

```
//
**************************************************************************
*
// ** Sonar Data Stream
// **
// ** Preliminary Specification
// ** Created by in alpabetical order:
// **       Chesapeake Technology, Inc.
// **       Marine Sonic Technology, Ltd.
// **       Other People Too Hopefully
// **
// ** This file describes an open standard for a Sonar Data Stream.
// ** The idea is to break up all of the data needed to be stored or
// ** transmitted into common groups.  Groups that do not need to be used
can
// ** be ignored if they exist in the data stream.
// **
// ** All data being recorded is time based so a large emphasis is placed
// ** on a synchronization packet being introduced into the data stream
// ** at regular intervals in order to keep all of the different data
// ** types synchronized.
// **
//
**************************************************************************
*
// ** - Version History -
// **
// ** APRIL 14, 2006
// ** V0.1
// ** - Created (Pre Release Version)
// ** APRIL 19, 2006
// ** V0.1.1
// ** - Added complex number types
// ** - Moved the Time Reference to the SYNC packet
// ** - Changed the dwMisc meaning from interval to additional SYNC Tag
// ** identifier (configured as a bit stream synchronization pattern).
// ** - Removed the version number from the end of the TAG
// ** V0.1.2
// ** - Modified the SDS sync packet header (moved the dwTag order)
// ** - Remove the *pData pointers in every packet that use them
// **   (sonar, strings, etc...) and added the description of the packet
and
// **   where to place the data.
// ** - Added fSOG to the NAV packet (was missing)
// ** - Added fSpeedOfSound to the SonarDataChannel structure.
// ** - Changed all DATASOURCE_FISH to DATASOURCE_SONAR because some
systems
// **   are fish less.
```

```
// **
// ** V0.1.3
// ** - Added a NMEA packet for storing NMEA strings received by the sonar
for
// **   later processing by the topside and debugging.  Also for packing
// **   Host Remote NMEA reply sentences into the Sonar Data Stream for use
// **   on towed systems that only have 1 communications channel that both
// **   command & data must ride on.
// **
// ** V0.1.4
// ** - Add the implied invalid value for data packets (99999.99)
// ** V0.9.0
// ** - Modify the SDS header for all packets except for SYNC packet
// **   to include a byte sized data checksum in the misc. bytes.  Also if
the
// **   data checksum (or header checksum is 0) then it should be ignored.
// ** - Add a new version of the Sonar Data packet.  This version combines
// **   more than 1 channel of data in one packet.  There is also the
option
// **   to split up left and right channel data into 2 packets using the
// **   odd/even data flag.
// ** - Consequently this version is nearing the release date of Sea Scan
HDS
// **   which will signal V1.0 of this data format.
// ** - Created a new data type for the wDataType SNR and SNR2 data types.
// **   12-bit unpacked, since we have used the previous 12 bit as packed
data.
// ** V0.10.0
// ** - Added the Layback Info data packet = LAY1
// ** V0.11.0
// ** - Added PORT2 and STARBOARD2 SONAR_TYPEs to support proper tagging
//      of simultaneous Low/High Frequency Side Scan Operation
//
*************************************************************************
*

//#include "standard.h"   // standard definitions & types

#ifndef __SDS_H__
#define __SDS_H__

// ****************************************


// Uncomment if this is supported by your compiler
// otherwise you must ensure that the structures are packed to 1 byte
boundaries
#pragma pack(push,1)
//#pragma pack(1)

// NOTE:    All data and variables are to be stored as "Little Endian" byte
order
//               Variable sizes are as follows:
```

```
//                BYTE        - 1 Byte
//                WORD        - 2 Bytes
//                DWORD       - 4 Bytes
//                int               - 4 Bytes
//                short int   - 2 Bytes
//                float       - 4 Bytes
//                double            - 8 Bytes

// - Packet Header -
// Each packet contains a header and data(optional)
// The header describes what the packet contains and its size in bytes

// NOTE:  Each packet must have the Size, Tag, and Timestamp filled out.
Misc. should
// be set to 0 unless otherwise noted.

// NOTE:  If there is a checksum defined then specifying 0 as the checksum
value should signal
// the SDS reader code to ignore the checksum value.

struct SDS_HEADER
{
      DWORD dwSize;       // Size of the Data Packet (not including the
header).
      DWORD dwTimeStamp;  // Time Stamp is the number of milliseconds since
Midnight GMT.
      DWORD dwTag;        // Tag describes the contents of the packet or
how the header will be used.
      BYTE  byMisc[3];    // Misc is for miscellaneous use and for data
that is less than 3 bytes.
      BYTE  byCheckSum;   // Simple 8 bit checksum.
};

// NOTE: Additional data packets can be defined.  Manufacturer specific
packets can be defined
// with a TAG beginning with 0x00###### which will give 24 bits worth of
choices.  Tags that are
// unknown to the reader should be ignored.
//

// - SYNC Packet -
// The SYNC packet allows the data stream to be sychronized
// it can be used to find a good starting point in corrupted data
// or it can be used to start a frame ... skipping to the next
// SYNC packet will advance the frame X amount of time
// The SYNC packets should be sent/formed at regular intervals
// specified in the packet.
//
// The SYNC packet is also used to Synchronize real time with the time
stamp time.
//
// NOTE:    Every file should have a SYNC packet at the start of the file.
//                If the SYNC packet is not at the beginning then a file
```

```
reader should advance to the
//                nearest SYNC packet and start reading from there.
//
// TAG:           0x53594E43 or ASCII "SYNC"
// MISC:    Contains a serial synchronization bit stream: 0xAAAAAA
//                Bit Stream: 10101010101010101010101010
//                Also serves the purpose of increasing the uniqueness of
the SYNC packet TAG
// DATA:    Contains a real time reference in order to synchronize the time
stamp in the header wiht
//                a real date and time.  The time and date contained is
referenced to GMT time.

#define SDS_TAG_SYNC    0x53594E43

struct SDS_DATA_SYNC
{
      DWORD dwReference;      // Numbers of seconds elapsed since midnight
(00:00:00), January 1, 1970
                                          // Most Windows and UNIX (POSIX)
based systems have the time() function
                                          // available.  The reference is
simply the output from time().
                                          // These systems normally carry
functions to convert this time reference
                                          // to many different formats.
      WORD  wInterval;        // Sync repition interval in milliseconds at
which the SYNC packet is sent.
                                          // The default interval is 1
second.
};

//
// - Sonar Data Packet -
// The sonar data packet allows the storage of multiple types of sonar
data.
// The sonar data starts with a SDS_SONAR_DATA_CHANNEL structure followed
by
// the actual sonar data itself.  Each individual channel has its own SDS
// Sonar Data packet.
//
// TAG: 0x534E5200 or ASCII "SNR"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

// - Sonar Types -
// Pick only 1 of these to go in the wSonarType variable field
//
// Custom and manufacturer specific sonar types begin at
SONARTYPE_USER_DEFINED

#define SDS_TAG_SNR          0x534E5200
```

```
#define SONARTYPE_PORT_SIDESCAN                 0x0001
#define SONARTYPE_STARBOARD_SIDESCAN            0x0002
#define SONARTYPE_PORT2_SIDESCAN                0x0010
#define SONARTYPE_STARBOARD2_SIDESCAN           0x0020
#define SONARTYPE_FORWARD                       0x0003
#define SONARTYPE_MULTIBEAM                     0x0004
#define SONARTYPE_SCANNING                      0x0005
#define SONARTYPE_SUBBOTTOM                     0x0006
#define SONARTYPE_USER_DEFINED                  0x0100

// - Data Types -
// Pick only 1 of these to go in the wDataType varaible field
//

// 8 Bit Data is stored 1 BYTE Wide
#define DATATYPE_UNSIGNED8BIT                   0x0001
#define DATATYPE_SIGNED8BIT                     0x0002

// 12 Bit Data Packed is stored 3 nibbles wide.  The high nibble of the 1st
sample should be the low nibble of the next byte for example).
#define DATATYPE_UNSIGNED12BIT_PACKED           0x0003
#define DATATYPE_SIGNED12BIT_PACKED             0x0004

// 12 Bit Data UnPacked is stored 2 BYTEs wide (the uppder nibble of th
high byte is 0).
#define DATATYPE_UNSIGNED12BIT_UNPACKED         0x000A
#define DATATYPE_SIGNED12BIT_UNPACKED           0x000B

// 16 Bit Data is stored 2 BYTEs Wide
#define DATATYPE_UNSIGNED16BIT                  0x0005
#define DATATYPE_SIGNED16BIT                    0x0006

// 32 Bit Data is stored 4 BYTEs Wide
#define DATATYPE_UNSIGNED32BIT                  0x000C
#define DATATYPE_SIGNED32BIT                    0x000D

// Floating Point is stored 4 BYTEs Wide and is a floating point number
available in 32 bit C++ compiler
#define DATATYPE_FLOAT                          0x0007

// Complex Integer is stored as a two consecutive signed 16 bit integers,
the real elemet first, followed by
// the imaginary element.
// For example: real0 complex0 real1 complex1 real2 complex2 ...
#define DATATYPE_COMPLEX_INT                    0x0008

// Complex Floating Point is stored as two consecutive floating point
numbers, the real element first
// , followed by the imaginary element.  See Complex Integer (above) for an
example.
#define DATATYPE_COMPLEX_FLOAT                  0x0009

struct SDS_DATA_SONAR_CHANNEL
```

```
{
      WORD  wSonarType;             // Type of sonar/transducer (Port,
Starboard, Forward Looking, Multibeam, Scanning, etc.)
      float fFreqHz;                // Acoustic frequency in Hz of this
channel
      float fSpeedOfSound;          // Speed of sound in water in meters
per second.
      float fRangeDelayMs;          // Delay from the start of the transmit
pulse to first sample in milliseconds
      float fRangeMs;               // Range in milliseconds that this set
of data represents
      BYTE  bySampleSizeBytes;      // Size in bytes of each sonar sample
      WORD  wDataType;              // Type of data stored in this data set
(UNSIGNED, SIGNED, 8 Bit, 12 Bit, 16 Bit, Complex, etc.)
      DWORD dwSamples;              // Number of samples in this channel
};

//
// - Sonar Data Packet Version 2-
// TODO - desription of sonar data packet version 2 and its special
features.
//
// TAG: 0x32524E52 or ASCII "SNR2"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

// - Sonar Types -
// Pick only 1 of these to go in the wSonarType variable field
// Please the SNR packet above for definitions for the wSonarType
//
// Custom and manufacturer specific sonar types begin at
SONARTYPE_USER_DEFINED

#define SDS_TAG_SNR2   0x32524E52

// - Data Types -
// Pick only 1 of these to go in the wDataType varaible field
// Please see the SNR packet above for definitions for the wDataType

// - Data Flags -
// Combine any of these flags to describe any additional features of the
data set contained within

#define DATAFLAG_NONE           0x0000      // Nothing special about this
data
#define DATAFLAG_RAWDATA        0x0001      // Raw, unprocessed sonar data
#define DATAFLAG_LOGCOMPRESSED  0x0002      // Logrithmically Compressed
Sonar Data
#define DATAFLAG_TVG            0x0004      // Sonar Data has had TVG
applied
#define DATAFLAG_EVENDATA       0x0008      // Sonar Data is the even data
from a ping (the ping is completed by the odd data)
#define DATAFLAG_ODDDATA        0x0010      // Sonar Data is the odd data
```

```
from a ping (the ping is completed by the even data)

struct SDS_DATA_SONAR_PING
{
  BYTE  byChannelCount;     // Number of channels in this ping
  DWORD dwPingNumber;       // Ping number for this sonar, incrementing
counter
  float fSpeedOfSound;      // Speed of sound in water in meters per second
  BYTE reserved[7];         // Reserved for later use (should be 0)
};

struct SDS_DATA_SONAR_PING_CHANNEL
{
  WORD  wSonarType;     // Type of sonar/transducer (Port, Starboard,
Forward Looking, Multibeam, Scanning, etc.)
  WORD  wSonarID;       // Transducer Identifier (I.E. Serial Number or
Position, if not used then set to 0)
  float fFreqHz;        // Acoustic frequency in Hz of this channel
  float fRangeMs;       // Range in milliseconds that this set of data
represents
  float fRangeDelayMs;  // Delay from the start of the transmit pulse to
first sample in milliseconds
  WORD  wDataFlags;     // Data Flags (additional information about data
samples see above for flags)
  WORD  wDataType;      // Type of data stored in this data set (UNSIGNED,
SIGNED, 8 Bit, 12 Bit, 16 Bit, Complex, etc.)
  WORD  wSamples;       // Number of samples in this channel
};

//
// - Sonar Data Packet Version 3-
// TODO - desription of sonar data packet version 2 and its special
features.
//
// TAG: 0x32524E53 or ASCII "SNR3"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

// - Sonar Types -
// Pick only 1 of these to go in the wSonarType variable field
// Please the SNR packet above for definitions for the wSonarType
//
// Custom and manufacturer specific sonar types begin at
SONARTYPE_USER_DEFINED

#define SDS_TAG_SNR3   0x32524E53

// - Data Types -
// Pick only 1 of these to go in the wDataType varaible field
// Please see the SNR packet above for definitions for the wDataType

// - Data Flags -
// Combine any of these flags to describe any additional features of the
```

```
data set contained within
// Please see the SNR2 packet above for definitions for the wDataFlags

struct SDS_DATA_SONAR3
{
  // Ping specific data
  DWORD dwPingNumber;   // Ping number for this sonar, incrementing counter
  float fSpeedOfSound;  // Speed of sound in water in meters per second
  WORD  wSonarType;     // Type of sonar/transducer (Port, Starboard,
Forward Looking, Multibeam, Scanning, etc.)
  WORD  wSonarID;       // Transducer Identifier (I.E. Serial Number or
Position, if not used then set to 0)
  float fFreqHz;        // Acoustic frequency in Hz of this channel
  float fRangeMs;       // Range in milliseconds that this set of data
represents
  float fRangeDelayMs;  // Delay from the start of the transmit pulse to
first sample in milliseconds
  WORD  wDataFlags;     // Data Flags (additional information about data
samples see above for flags)
  WORD  wDataType;      // Type of data stored in this data set (UNSIGNED,
SIGNED, 8 Bit, 12 Bit, 16 Bit, Complex, etc.)

  // Packet specific data
  WORD  wPacketNumber;  // Packet number for this ping
  WORD  wSampleStart;   // Start sample number
  WORD  wSamples;       // Number of samples in this packet
};

// - Sources for the Data Packets that contain the variable wSource -
//

#define DATASOURCE_SHIP                   0x0001
#define DATASOURCE_SONAR                  0x0002
#define DATASOURCE_SENSOR1                0x0003
#define DATASOURCE_SENSOR2                0x0004
#define DATASOURCE_SENSOR3                0x0005
#define DATASOURCE_SENSOR4                0x0006

// Invalid Values for the Data Packets
#define INVALID_FLOAT_VALUE               99999.99F

// - Navigation Data Packet -
// The navigation data packet allows the storage of navigation data
//
// TAG: 0x4E415600 or ASCII "NAV"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_NAV          0x4E415600

struct SDS_DATA_NAV
{
      WORD  wSource;          // Origin of this navigation data (Ship,
```

```
Towfish, etc.)
      double      dLatitude;        // Latitude ecoded as decimal degrees
      double      dLongitude;       // Longitude encoded as decimal degrees
      float fCOG;             // Course Over Ground in degrees (true NOT
magnetic)
      float fHeading;         // Heading in degrees (true NOT magnetic)
      float fSOG;             // Speed Over Ground (in meters per second)
};

// - Orientation Data Packet -
// The orientation from the origin of the navigation data
//
// TAG: 0x4F524E54 or ASCII "ORNT"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_ORNT    0x4F524E54

struct SDS_DATA_ORIENTATION
{
      WORD  wSource;          // Origin of the navigation data (Ship,
Towfish, etc.)
      float fX;                     // X offset in meters from the origin
      float fY;                     // Y offset in meters from the origin
      float fZ;                     // Z offset in meters from the origin
      float fRoll;                  // Roll in degrees
      float fPitch;                 // Pitch in degrees
      float fYaw;             // Yaw in degrees
      float fHeave;                 // Heave in cm
};

// - Fathometer Data Packet -
// The depth and altitude information of the origin of the data
//
// TAG:     0x46415400 or ASCII "FAT"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_FAT           0x46415400

struct SDS_DATA_FATHOMETER
{
      WORD  wSource;          // Origin of the fathometer data (Ship,
Towfish, etc.)
      float fDepth;                 // Depth in meters
      float fAltitude;        // Altitude in meters
};

// - Magnetometer Data Packet -
// The reading from a magnetometer sensor
//
// TAG:     0x4D414700 or ASCII "MAG"
```

```
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)
// 1 Gamma = 1/100000 Gauss or 1/100 milliGauss????
#define SDS_TAG_MAG           0x4D414700

struct SDS_DATA_MAGNETOMETER
{
      WORD  wSource;                    // Origin of the magnetometer
data (Ship, Towfish, etc.)
      float fXReading;                  // Magnetometer reading in Gammas
- X Axis
      float       fYReading;                  // Magnetometer reading in
Gammas - Y Axis
      float fZReading;                  // Magnetometer reading in Gammas
- Z Axis
};

// - Marker/Annotation Data Packet -
// The marker annotation data packet contains a description string as well
as location in Lat/Lon
// and offset information.
// The marker annotation data packet begins with a SDS_DATA_MARK structure
and is followed by an array of
// ASCII characters that form the description string.  The string can be a
maximum of 1024 characters
// including the NULL terminator.
//
// TAG: 0x4D41524B or ASCII "MARK"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_MARK    0x4D41524B

struct SDS_DATA_MARK
{
      double      dLat;             // Origin Latitude of the Mark
      double      dLon;             // Origin Longitude of the Mark
      float fXOffset;         // Offset in meters in the X direction from
the origin
      float fYOffset;         // Offset in meters in the Y direction from
the origin
      float fZOffset;         // Offset in meters in the Z direction from
the origin
      WORD  wLength;          // Length of the string in characters
including the NULL terminator (maximum 1024)
};

// - NMEA Sentece Data Packet -
// The NMEA sentence data packet contains NMEA formatted strings that are
received and
// sent by the sonar or device that is generating the Sonar Data Stream.
This can be
// used for later processing by the topside and debugging.  Also for
```

```
packing Host Remote
// NMEA reply sentences into the Sonar Data Stream for use on towed systems
that have
// only 1 communications channel that both command & data must ride on.
The NMEA sentence
// data packet begins with the SDS_DATA_NMEA structure and is follwed by an
array of ASCII
// characters that form the NMEA string.  The string can be a maximum of
1024 characters
// including the NULL terminator.
//
// TAG: 0x4E4D4541 or ASCII "NMEA"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_NMEA    0x4E4D4541

#define DATASOURCE_SHIP                     0x0001
#define DATASOURCE_SONAR                    0x0002
#define DATASOURCE_SENSOR1                 0x0003
#define DATASOURCE_SENSOR2                 0x0004
#define DATASOURCE_SENSOR3                 0x0005
#define DATASOURCE_SENSOR4                 0x0006

struct SDS_DATA_NMEA
{
      WORD wSource;                 // Source of the NMEA sentence
      WORD wLength;                 // Length of the string in characters
including the NULL terminator (maximum 1024)
};

// - Cable Out Data Packet -
//
// TAG: 0x434F5431 or ASCII "COT1"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_COT1    0x434F5431

struct SDS_DATA_COT1
{
      // Cable out is ALWAYS from the ship
      double      dCableOut;              // Cable out in meters
};

// - Layback Data Packet -
// The layback information of the SONAR
//
// TAG:     0x4C415931 or ASCII "LAY1"
// MISC: 0, 0, 8 bit data checksum (a sum of all of the bytes in the data
payload)

#define SDS_TAG_LAY1    0x4C415931
```

```
struct SDS_DATA_LAYBACK1
{
      WORD  wSource;                // Origin of the Navigation data this
data modifies (Ship, Towfish, etc.)

      double      dOffsetX;               // Towpoint Offset from the
navigation origin - this modifies the navigation
                                          // for the navigation
originating from the wSource
                                          // + Is towards the
STARBOARD, - is towards the PORT
      double      dOffsetY;               // Towpoint Offset from the
navigation origin
                                          // + Is towards BOW, - is
towards STERN
      double      dOffsetZ;               // Towpoint Offset from the
navigation origin
                                          // + Is UP, - is DOWN

      double      dCableOut;              // Cable out in meters

      double      dCatenaryAdjustment;// Percentage of cable out to use in
calculation
                                          // due to cable catenary
(due to cable/sonar fluid resistance).
                                          // This can be a value from
0.0% to 100.0% - typcial value is 95%
                                          // Or in other words a 5%
length loss due to the catenary.
                                          // The percentage should be
stored as a value from 0.0 to 1.0.

      double      dWaterOffset;           // Offset of the towpoint to the
water in meters (depth offset)
                                          // This is a positive real
number. 0 means no offset.  This value
                                          // Essentially adds to the
water depth value.

      double      dDepthBelowWater; // Depth below the water in meters
                                          // Where 0 = water line, >
0 is under water
};

// Include some custom SDS data packets here
#include "mstl_custom_sds.h"

// Uncomment if this is supported by your compiler
// otherwise you must ensure that the structures are packed to 1 byte
boundaries
#pragma pack(pop)
```

```
//#pragma pack()

#endif      // __SDS_H__
```

3.2 Contact Information

For technical support please contact us using one of the methods listed below:

Phone: (804) 693-9602
Toll Free: (800) 447-4804
Website: www.marinesonic.com

Business Hours: 8:00 AM to 5:00 PM EST

Additional documentation can be found on our Sea Scan HDS Documentation page which can be found by going to our web site and following the links:
Downloads -> Sea Scan HDS -> Documents

Index

- S -

Sonar Data Packet Data Types 9