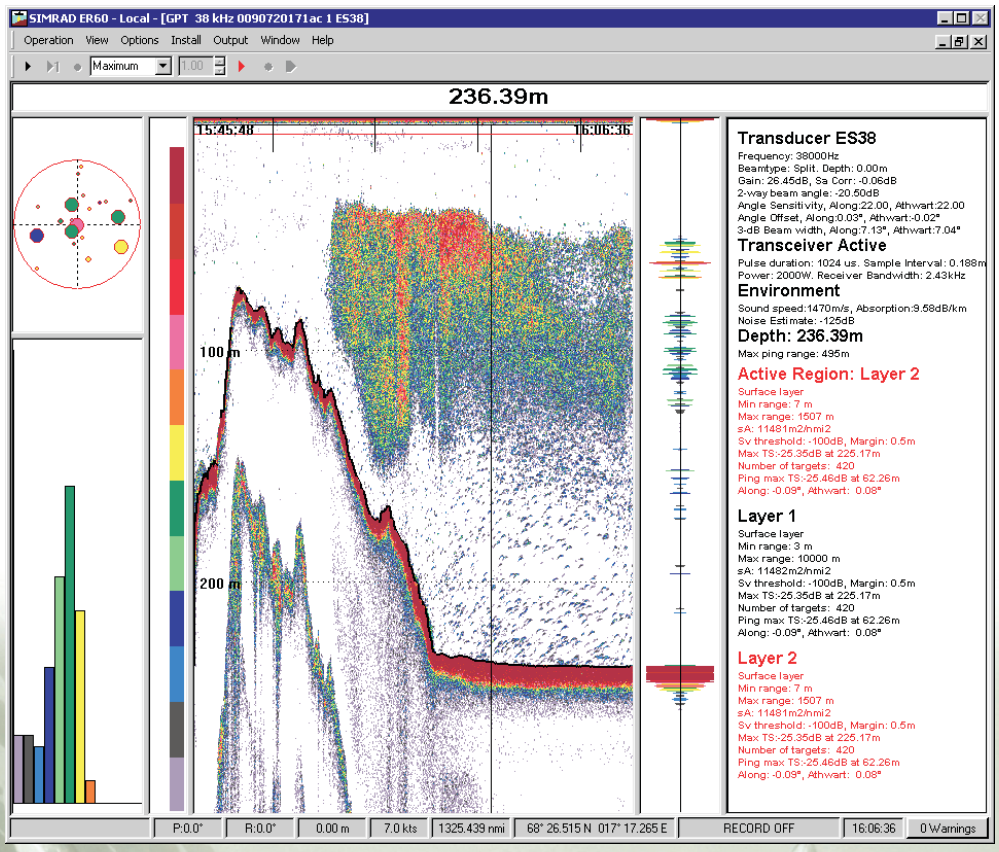
Reference manual

Simrad EK60

Scientific echo sounder



KONGSBERG

Simrad EK60

Reference Manual

Release 2.4.X

This reference manual describes how to use the Simrad EK60 and EY60 scientific echo sounders. The software application used to operate and control the EK60 and EY60 systems is referred to as Simrad ER60.

164692/D

Revision status

Document number: 164692 /Revision D / ISBN-13: 978-8066-011-4		
Rev.A	July 2003	First version.
Rev.B	September 2004	Revised for software version 2.1.0.
Rev.C	January 2008	Revised for software version 2.2.0. Document transferred to XML format.
Rev.D	February 2012	Revised for software version 2.4.x. The product name changed from ER60 to EK60.

Disclaimer

Kongsberg Maritime AS endeavours to ensure that all information in this document is correct and fairly stated, but does not accept liability for any errors or omissions.

Warning

The equipment to which this manual applies must only be used for the purpose for which it was designed. Improper use or maintenance may cause damage to the equipment and/or injury to personnel. All users must be familiar with the contents of the appropriate manuals before attempting to install, operate, maintain or in any other way work on the equipment. Kongsberg Maritime AS disclaims any responsibility for damage or injury caused by improper installation, use or maintenance of the equipment.

Support information

If you require maintenance or repair, contact your local dealer. You can also contact us using the following address: simrad.support@simrad.com. If you need information about our other products, visit http://www.simrad.com. On this website you will also find a list of our dealers and distributors.
See also *Support information* on page 16.

Table of contents

About this manual

Purpose

The purpose of this reference manual is to provide the descriptions, procedures and detailed parameter explanations required to allow for safe and efficient use of the Simrad EK60, as well as a thorough understanding of the system parameters and adjustments.

A good understanding of system functions and controls is essential to fully take advantage of the functionality provided. Sea conditions vary, sometimes drastically, and it is not possible to identify settings that will provide the best data at all times. Careful study of the information in this manual is highly recommended, preferably while exploring the system's functionality.

Target audience

The manual is intended for all users of the Simrad EK60. Due to the nature of the descriptions and the level of detail provided by this manual, it is well suited for those who are – or wish to be – expert users.

We assume that you are familiar with the basic acoustic principles of sound in water, and that you have some experience with echo sounder operation.

Click "Help"!

Installed on your Simrad EK60 you will find a comprehensive on-line help system. You may not find it in your language, but everything you can read in the *Simrad EK60 Reference manual* can also be found in the context sensitive on-line help.

To access this information click **Help** on the **Main** menu, or the **Help** button in one of the dialogs.

Note that when you open the help system it will place itself on the top of the display presentation!

Online information

All documentation provided for your Simrad EK60 can be downloaded from http://www.simrad.com.

Software version

This manual complies to software version 2.4.X.

Registered trademarks

Windows®, Windows® XP®, and Windows® 7 are either registered trademarks, or trademarks of Microsoft Corporation in the United States and/or other countries.

Simrad®, SIMRAD® and the Simrad® logo are either registered trademarks, or trademarks of Kongsberg Maritime AS in Norway and other countries.

Telegram formats

This chapter details the standard NMEA, third party and proprietary Simrad telegrams, as well as – if applicable – dedicated file formats for data transfer.

According to their web site, the *National Marine Electronics Association (NMEA)* is the unifying force behind the entire marine electronics industry, bringing together all aspects of the industry for the betterment of all in our business.

All NMEA, third party and proprietary telegrams available are not described here, only those used by the EK60. If the specifications here differ from the original specifications published by NMEA, the specifications issued by NMEA must be regarded as the correct version.

Topics

Related topics

- http://www.nmea.org.

About the NMEA telegram format

The Simrad EK60 can send and receive information to and from several different peripherals. All transmissions take place as **telegrams** with data sentences. Each telegram has a defined format and length.

The **NMEA 0183** standard is the most common protocol used to receive and transmit data to and from peripheral sensors. A parametric sentence structure is used for all NMEA data. The sentence start with a "$" delimiter, and represent the majority of approved sentences defined by the standard. This sentence structure, with delimited and defined data files, is the preferred method for conveying information.

For more information about the NMEA standard, the format and the data sentences, refer to their official publications. Their document *NMEA 1083 - Standard for interfacing marine electronic devices* explains the formats in detail. The document can be obtained from http://www.nmea.org.

Topics

National Marine Electronics Association (NMEA)

The *National Marine Electronics Association (NMEA)* has defined communication standards for maritime electronic equipment, and the EK60 echo sounder conforms to these standards. The most common standard is *NMEA 0183*, and the National Marine Electronics Association describes it as follows:

> The NMEA 0183 Interface Standard defines electrical signal requirements, data transmission protocol and time, and specific sentence formats for a 4800 baud serial data bus. Each bus may have only one talker but many listeners.
>
> — *National Marine Electronics Association*

For more information about the National Marine Electronics Association and the NMEA 0183 standard, refer to the organization's web site at http://www.nmea.org.

NMEA telegram principles

To move information between two electronic units, the data are collected in **telegrams**. The content (protocol) of each telegram is defined by the NMEA standard, and several telegram types exist to allow different type of data to be distributed.

The phrase **datagram** is also frequently used about this communication method.

Unless you wish to write your own software, you do not need to know how these telegrams are designed. However, whenever you set up equipment interfaces, you need to ensure that each system on your communication line is set up to send and receive the same telegram. The standard allows one system to send data (a “talker") and several others to receive data simultaneously ("listeners") on the same line. Therefore, you must ensure that all products receiving data on a communication line is set up to receive the same telegram(s) that the transmitting product provides.

Standard NMEA 0183 communication parameters

The communication parameters defined for **NMEA 0183** are:

- **Baudrate**: 4800 bits per second
- **Data bits**: 8
- **Parity**: None
- **Stop bits**: One

Some instruments will also offer other parameters and/or choices.

NMEA sentence structure

The following provides a summary explanation of the approved parametric sentence structure.

`$aaccc,c—c*hh<CR><LF>`

1 **"$"**: *Start of sentence* (Hex: 24).

2 **aaccc**: *Address field*. The first two characters (**aa**) identifies the *Talker ID*, while the last three characters are the *Sentence formatter* mnemonic code identifying the data type and the string format of the successive fields.

3 **","**: *Field delimiter* (Hex: 2C). This character starts each field except the address and checksum fields. If it is followed by a null field, it is all that remains to indicate no data in the field.

4 **c—c**: *Data sentence block*. This is a series of data fields containing all the data to be transmitted. The data field sentence is fixed and identified by the *Sentence formatter* in the address field. Data fields may be of variable lengths, and they are preceded by the *Field delimiter*.

5 **"*"**: *Checksum delimiter* (Hex: 2A). This delimiter follows the last field of the sentence, and indicates that the following two alphanumerical characters contain the checksum.

6 **hh**: *Checksum*

7 **<CR><LF>**: *Terminates sentence*

Proprietary telegrams

In some proprietary telegrams received from other Simrad equipment, the **$** character is replaced by the **@** character. The checksum field may then not be in use.

Specification of NMEA telegrams

All standard NMEA telegrams supported by the EK60 are specified here. The information is extracted from the original NMEA specifications. If additional details about the individual telegram formats are required, see the original source file.

Topics

DBS Depth below surface

This telegram provides the current depth from the surface. The telegram is no longer recommended for use in new designs.

It is often replaced by the **DPT** telegram.

Format

```
$--DBS,x.x,f,y.y,M,z.z,F*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **DBS** = telegram identifier

3 **x.x,f** = depth below surface in feet

4 **y.y,M** = depth below surface in meters

5 **z.z,F** = depth below surface in fathoms

Related topics

- *Depth Output* on page 128

DBT Depth below transducer

This telegram provides the water depth referenced to the transducer.

Format

```
$--DBT,x.x,f,y.y,M,z.z,F*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **DBT** = telegram identifier

3 **x.x,f** = water depth below transducer in feet

4 **y.y,M** = water depth below transducer in meters

5 **z.z,F** = water depth below transducer in fathoms

Related topics

- *Depth Output* on page 128

DPT Depth

This telegram contains water depth relative to the transducer and offset of the measuring transducer. Positive offset numbers provide the distance from the transducer to the water line. Negative offset numbers provide the distance from the transducer to the part of the keel of interest.

For additional details, refer to the NMEA standard.

Format

```
$--DPT,x.x,y.y,z.z*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **DPT** = telegram identifier

3 **x.x** = water depth, in meters, relative to the transducer

4 **y.y** = offset, in meters, from the transducer

5 **z.z** = maximum range scale in use

Related topics

- *Depth Output* on page 128

GGA Global positioning system fix data

This telegram contains time, position and fix related data from a global positioning system (GPS).

Format

```
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,
x,zz,d.d,a.a,M,g.g,M,r.r,cccc*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **GGA** = telegram identifier

3 **hhmmss.ss** = coordinated universal time (UTC) of position

4 **llll.ll** = latitude north/south, position in degrees, minutes and hundredths.

5 **a** = North/South. Characters **N** (North) or **S** (South) identifies the bearing.

6 **yyyyy.yy** = longitude east/west, position in degrees, minutes and hundredths.

7 **a** = West/East. Characters **W** (West) or **E** (East) identifies the bearing.

8 **x** = GPS quality indicator (refer to the NMEA standard for further details)

9 **zz** = number of satellites in use, 00 to 12, may be different from the number in view

10 **d.d** = horizontal dilution of precision

11 **a.a,M** = altitude related to mean sea level (geoid) in meters

12 **g.g,M** = geoidal separation in meters

13 **r.r** = age of differential GPS data

14 **cccc** = differential reference station identification, 0000 to 1023

Related topics

- *Navigation; Position tab* on page 102

GLL Geographical position latitude/longitude

This telegram is used to transfer latitude and longitude of vessel position, time of position fix and status from a global positioning system (GPS).

Format

```
$--GLL,llll.ll,a,yyyyy.yy,a,
hhmmss.ss,A,a*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **GLL** = telegram identifier.

3 **llll.ll** = latitude north/south, position in degrees, minutes and hundredths. Characters **N** (North) or **S** (South) identifies the bearing.

4 **a** = North/South. Characters **N** (North) or **S** (South) identifies the bearing.

5 **yyyyy.yy,a** = longitude east/west, position in degrees, minutes and hundredths.

6 **a** = West/East. Characters **W** (West) or **E** (East) identifies the bearing.

7 **hhmmss.ss** = coordinated universal time (UTC) of position.

8 **A** = status, characters **A** (data valid) or **V** (data not valid) are used.

9 **a** = mode indicator.

Related topics

- *Navigation; Position tab* on page 102

HDG Heading, deviation and variation

This telegram contains the heading from a magnetic sensor, which if corrected for deviation will produce magnetic heading, which if offset by variation will provide true heading.

Format

```
$--HDG,x.x,z.z,a,r.r,a*hh<CR><LF>
```

Heading conversions

To obtain magnetic heading: Add easterly deviation (E) to magnetic sensor reading, or subtract westerly deviation (W) from magnetic sensor reading.

To obtain true heading: Add easterly variation (E) to magnetic heading, or subtract westerly variation (W) from magnetic heading.

Format description

1 – – = talker identifier

2 **HDG** = telegram identifier

3 **x.x** = magnetic sensor heading, degrees

4 **z.z,a** = magnetic deviation, degrees east/west

5 **r.r,a** = magnetic variation, degrees east/west

Related topics

- *Navigation; Heading tab* on page 106

HDM Heading, magnetic

This telegram contains vessel heading in degrees magnetic. The telegram is no longer recommended for use in new designs.

It is often replaced by the **HDG** telegram.

Format

```
$--HDM,x.x,M*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **HDM** = telegram identifier

3 **x.x** = heading in degrees, magnetic

Related topics

- *Navigation; Heading tab* on page 106

HDT Heading, true

This telegram is used to transfer heading information from a gyro.

Format

```
$--HDT,x.x,T*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **HDT** = telegram identifier

3 **x.x,T** = heading, degrees true

Related topics

- *Navigation; Heading tab* on page 106

RMC Recommended minimum specific GNSS data

This telegram contains time, date, position, course and speed data provided by a global navigation satellite system (GNSS) receiver.

Format

```
$--RMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,
x.x,z.z,ddmmyy,r.r,a,a*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **RMC** = telegram identifier

3 **hhmmss.ss** = coordinated universal time (UTC) of position fix

4 **A** = status, characters **A** (data valid) or **V** (Navigation receiver warning) are used.

5 **llll.ll,a** = latitude nort/south. Characters **N** (North) or **S** (South) identifies the bearing.

6 **yyyyy.yy.a** = longitude east/west. Characters **E** (East) or **W** (West) identifies the bearing.

7 **x.x** = speed over ground, knots

8 **z.z** = course over ground, degrees true

9 **ddmmyy** = date

10 **r.r,a** = magnetic variation, degrees east/west. Characters **E** (East) or **W** (West) identifies the bearing.

11 **a** = mode indicator

Related topics

- *Navigation; Position tab* on page 102
- *Navigation; Speed tab* on page 103

VBW Dual ground and water speed

This telegram contains water referenced and ground referenced speed data.

Format

```
$--VBW,x.x,z.z,A,r.r,q.q,A,p.p,A,
c.c,A*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **VBW** = telegram identifier

3 **x.x** = longitude water speed, knots

4 **z.z** = transverse water speed, knots

5 **A** = status, water speed, characters **A** (data valid) or **V** (data not valid) are used.

6 **r.r** = longitudinal ground speed, knots

7 **q.q** = transverse ground speed, knots

8 **A** = status, ground speed, characters **A** (data valid) or **V** (data not valid) are used.

9 **p.p** = stern transverse water speed, knots

10 **A** = status, stern water speed, characters **A** (data valid) or **V** (data not valid) are used.

11 **c.c** = stern transverse ground speed, knots

12 **A** = status, stern ground speed, characters **A** (data valid) or **V** (data not valid) are used.

Related topics

- *Navigation; Speed tab* on page 103

VHW Water speed and heading

This telegram contains the compass heading to which the vessel points and the speed of the vessel relative to the water.

Format

```
$--VHW,x.x,T,x.x,M,x.x,N,x.x,K*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **VHW** = telegram identifier

3 **x.x,T** = heading, degrees true

4 **x.x,M** = heading, degrees magnetic

5 **x.x,N** = speed relative to water, knots, resolution 0.1

6 **x.x,K** = speed relative to water, km/hr, resolution 0.1

Related topics

- *Navigation; Heading tab* on page 106
- *Navigation; Speed tab* on page 103

VLW Dual ground/water distance

This telegram contains the distance travelled relative to the water and over the ground.

Format

```
$--VLW,x.x,N,y.y,N,z.z,N,g.g,N*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **VLW** = telegram identifier

3 **x.x,N** = total cumulative water distance, nautical miles.

4 **y.y,N** = water distance since reset, nautical miles.

5 **z.z,N** = total cumulative ground distance, nautical miles.

6 **g.g,N** = ground distance since reset, nautical miles.

Related topics

- *Navigation; Distance tab* on page 104

VTG Course over ground & ground speed

This telegram contains the actual course and speed relative to the ground.

Format

```
$--VTG,x.x,T,y.y,M,z.z,N,g.g,K,a*hh<CR><LF>
```

Format description

1 – – = talker identifier

2 **VTG** = telegram identifier

3 **x.x,T** = course over ground, degrees true

4 **y.y,M** = course over ground, degrees magnetic

5 **z.z,N** = speed over ground, knots, resolution 0.1

6 **g.g,K** = speed over ground, km/hr, resolution 0.1

7 **a** = mode indicator

Related topics

- *Navigation; Distance tab* on page 104

Proprietary telegrams and formats

These are the proprietary telegrams supported by the EK60. These telegram formats have all been defined by Simrad. The telegrams are listed in alphabetical order.

Topics

Simrad EK500 Depth telegram

This proprietary Simrad telegram was defined for the EK500 scientific echo sounder. It provides the current depth from three channels, as well as the bottom surface backscattering strength and the athwartships bottom slope. This telegram has be defined for output on either a serial line or a local area network Ethernet connection.

Serial line format

```
D#,hhmmsstt,x.x,y.y,t,s.s<CR><LF>
```

Serial line format description

1 **D#** = identifier, can be **D1**, **D2** or **D3** for channels 1, 2 or 3.

2 **hhmmsstt** = current time; hour, minute, second and hundredth of second

3 **x.x** = detected bottom depth in meters

4 **y.y** = bottom surface backscattering strength in dB

5 **t** = transducer number

6 **s,s** = athwartships bottom slope in degrees

Ethernet format

The Ethernet line output is specified using a “C” programming language structure. Note that this format does not include carriage return and line feed characters at the end of the telegram.

```
struct Depth {
  char Header[2];
  char Separator1[1];
  char Time[8];
  char Separator1[2];
  float Depth[4];
  float Ss[4];
```

```
  long TransducerNumber[4];
  float AthwartShips;
};
```

Ethernet format description

1 **Header#** = can be **D1**, **D2** or **D3** for channels 1, 2 or 3.

2 **Separator** = ","

3 **Time** = current time; hour, minute, second and hundredth of second

4 **Depth** = detected bottom depth in meters

5 **Ss** = bottom surface backscattering strength in dB

6 **TransducerNumber** = transducer number

7 **AthwartShips** = athwartships bottom slope in degrees

Kongsberg EM Attitude 1000

This proprietary **Kongsberg EM Attitude 1000** binary telegram consists of a fixed length message with 10 bytes.

It is defined as follows:

- Byte 1: Sync byte 1 = 00h
- Byte 2: Sync byte 2 = 90h
- Byte 3: Roll LSB
- Byte 4: Roll MSB
- Byte 5: Pitch LSB
- Byte 6: Pitch MSB
- Byte 7: Heave LSB
- Byte 8: Heave MSB
- Byte 9: Heading LSB
- Byte 10: Heading MSB

LSB = least significant byte, MSB = most significant byte.

1 All data are in 2's complement binary, with 0.01° resolution for roll, pitch and heading, and 1 cm resolution for heave.
   - Roll is positive with port side up with ±179.99° valid range
   - Pitch is positive with bow up with ±179.99° valid range
   - Heave is positive up with ±9.99 m valid range
   - Heading is positive clockwise with 0 to 359.99° valid range

2 Non-valid data are assumed when a value is outside the valid range.

3 You can define how roll is assumed to be measured, either with respect to the horizontal plane (the Hippy 120 or TSS convention), or to the plane tilted by the given pitch angle (i.e. as a rotation angle around the pitch tilted forward pointing x-axis). The latter convention (called Tate-Bryant in the POS/MV documentation)

is used inside the system in all data displays and in logged data (a transformation is applied if the roll is given with respect to the horizontal).

4 Note that heave is displayed and logged as positive downwards (the sign is changed). Heave is corrected for roll and pitch.

5 This format was originally designed for use with the EM 950 and the EM 1000 multibeam echo sounders with the first synchronisation byte always assumed to be zero. The sensor manufacturers was then requested to include sensor status in the format using the first synchronisation byte for this purpose. With this additional information added, the datagram format is known as **Kongsberg EM Attitude 3000**.

Kongsberg EM Attitude 3000

This proprietary Kongsberg binary telegram consists of a fixed length 10-bytes message.

It is defined as follows:

- Byte 1: Sync byte 1 = 00h, or Sensor status = 90h-AFh
- Byte 2: Sync byte 2 = 90h
- Byte 3: Roll LSB
- Byte 4: Roll MSB
- Byte 5: Pitch LSB
- Byte 6: Pitch MSB
- Byte 7: Heave LSB
- Byte 8: Heave MSB
- Byte 9: Heading LSB
- Byte 10: Heading MSB

LSB = least significant byte, MSB = most significant byte.

1 All data are in 2's complement binary, with 0.01° resolution for roll, pitch and heading, and 1 cm resolution for heave.

- Roll is positive with port side up with ±179.99° valid range
- Pitch is positive with bow up with ±179.99° valid range
- Heave is positive up with ±9.99 m valid range
- Heading is positive clockwise with 0 to 359.99° valid range

Non-valid data are assumed when a value is outside the valid range.

2 You can define how roll is assumed to be measured, either with respect to the horizontal plane (the *Hippy 120* or *TSS* convention), or to the plane tilted by the given pitch angle (i.e. as a rotation angle around the pitch tilted forward pointing x-axis). The latter convention (called *Tate-Bryant* in the *POS/MV* documentation) is used inside the system in all data displays and in logged data (a transformation is applied if the roll is given with respect to the horizontal).

3 Note that heave is displayed and logged as positive downwards (the sign is changed) including roll and pitch induced lever arm translation to the system's transmit transducer.

4 This format has previously been used with the EM 950 and the EM 1000 with the first synchronisation byte always assumed to be zero (Datagram “Kongsberg EM Attitude 1000”). The sensor manufacturers have been requested to include sensor status in the format using the first synchronisation byte for this purpose.

It is thus assumed that:

- **90h** in the first byte indicates a valid measurement with full accuracy
- any value from **91h** to **99h** indicates valid data with reduced accuracy (decreasing accuracy with increasing number)
- any value from **9Ah** to **9Fh** indicates non-valid data but normal operation (for example configuration or calibration mode)
- and any value from **A0h** to **AFh** indicates a sensor error status

DBS Depth of trawl below surface

This proprietary Simrad telegram contains the depth of the trawl sensor.

Format

```
@IIDBS,,,x.x,M,,<CR><LF>
```

Format description

1 **II** = talker identifier (mandatory)

2 **DBS** = telegram identifier

3 **x.x,M** = depth in meters (0 to 2000)

HFB Trawl headrope to footrope and bottom

This proprietary Simrad telegram contains the distance from the headrope to the footrope, and from the footrope to the bottom.

Format

```
@IIHFB,x.x,M,y.y,M<CR><LF>
```

Format description

1 **II** = talker identifier (mandatory)

2 **HFB** = telegram identifier

3 **x.x,M** = distance from headrope to footrope, meters

4 **y.y, M** = distance from footrope to bottom, meters

PSIMP-D PI Sensor data

This proprietary Simrad telegram contains the type and configuration of PS and PI sensors used by the external PI catch monitoring system.

Note

This description is not complete. For further information, contact Simrad.

Format

```
$PSIMP,D,tt,dd,M,U,S,C,V,Cr,Q,In,SL,NL,G,
Cb,error*chksum<CR><LF>
```

Format description

1 **PS** = Talker identifier (mandatory)

2 **IMP** = Telegram identifier

3 **D** = Sentence specifier

4 **tt** = Time of day

5 **dd** = Current date

6 **M** = Measurement type:

- D = Depth
- T = Temperature
- C = Catch
- B = Bottom
- N = No sensor
- M = Marker

7 **U** = unit; M, f or F for depth measurements, C or F for temperature measurements

8 **S** = source; number (1, 2 or 3) of the sensor providing the current data values

9 **C** = channel; the number (1 to 30) of the communication channel for the current data source

10 **V** = value; the magnitude of the current sensor measurement

11 **Cr** = change rate; the magnitude of the current depth or temperature measurement

12 **Q** = quality:

- 0 = No connection between the sensor and the receiver
- 1 = One or two telemetry pulses are lost, current value is predicted
- 2 = The current data value is reliable

13 **In** = interference:

- 0 = No interference
- 1 = Interference detected

14 **SL** = signal level – the signal level of the telemetry pulse, measured in dB // 1 µPa

15 **NL** = noise level – the average noise level of the current channel, measured in dB // 1 µPa

16 **G** = the current gain; 0, 20 or 40 dB.

17 **Cb** = cable quality:

- 0 = cable is not connected
- 1 = cable is OK
- 2 = a short circuit, or the hydrophone current is too large

18 **error** = error detected – 0 when no error is detected, a number >0 indicates an error condition

19 **chksum** = The checksum field consists of a "*" and two hex digits representing the exclusive OR of all characters between, but not including, the "$" and "*" characters

PSIMDHB Bottom hardness and biomass

This proprietary Simrad telegram contains the bottom hardness and biomass as calculated by an echo sounder.

Format

```
$PSIMDHB,hhmmss.ss,t,f,KHZ,x.x,M,y.y,DB,z.z,,,<CR><LF>
```

Format description

1 **$P** = talker identifier (mandatory)

2 **SIM** = Simrad talker ID

3 **DHB** = coordinated universal time (UTC)

4 **hhmmss.ss** = time

5 **t** = transducer number

6 **f,KHZ** = echo sounder frequency in kHz

7 **x.x,M** = detected bottom depth in meters. Given as DBS (depth below surface), assuming proper transducer draft has been entered.

8 **y.y,DB** = bottom surface hardness in dB

9 **z.z** = relative biomass density in m²/nmi² (NASC) (s_A)

NASC means Nautical Area Scattering Coefficient. This is the format (s_A m²/nmi²) we provide the biomass data.

10 **spare1** = spare for future expansions

11 **spare2** = spare for future expansions

Simrad Sounder/TSS1 Motion protocol

This proprietary **Simrad Sounder/TSS1** protocol may be the most common interface for heave, roll and pitch compensation. When you select this protocol, the number of sensor variables is fixed, and there is no token associated with it. However, baud rate and output rate may be adjusted to fit your needs. The format is based on ASCII characters, the datagrams have fixed length, and it is terminated with a carriage return and line feed.

The definition of the attitude angles in this format is different from the *Euler* angles definition used elsewhere. The difference appears in the roll angle, where:

$$Roll_{echosounder} = \arcsin\left[\sin(Roll_{Euler}) \bullet \cos(Pitch_{Euler})\right]$$

Format

```
:aabbbb shhhhxsrrrr spppp<cr><lf>
```

Format description

1 **aa** = sway – two characters hex number with sway acceleration, in 0.03835 m/ss units

2 **bbbb** = heave – four characters hex number with heave acceleration, in 0.000625 m/ss units

3 **s** = a single character providing a “space” character if the value is positive, or a “–” character if it is negative

4 **hhhh** = heave – four characters decimal number with heave position in centimetres, positive up

5 **x** = status character:

- **U** = Unaided mode and stable data. The sensor operates without external input data.
- **u** = Unaided mode but unstable data. The sensor is without external input data, but the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on restarted. The alignment period from a power recycle is normally approximately five minutes.
- **G** = Speed aided mode and stable data. The sensor operates with external input of speed data.
- **g** = Speed aided mode but unstable data. The sensor operates with external input of speed data, but the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on restarted, or a failure in the speed data input.
- **H** = Heading aided mode and stable data. The sensor operates with external input of heading data.
- **h** = Heading aided mode but unstable data. The sensor operates with external input of heading data, but the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on restarted, or a failure in the heading data input.
- **F** = Full aided mode and stable data. The sensor operates with external input of both speed and heading data.
- **f** = Full aided mode but unstable data. The sensor operates with external input of heading and speed data, but the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on restarted, or a failure in the heading and/or speed data input.

6 **s** = a single character providing a “space” character if the value is positive, or a “–” character if it is negative

7 **rrrr** = roll – four character decimal number with roll angle in hundreds of a degree

8 **s** = a single character providing a “space” character if the value is positive, or a “–” character if it is negative

9 **pppp** = pitch – four character decimal number with pitch angle in hundreds of a degree

Simrad ATS Annotation

This proprietary Simrad telegram contains a text string to be used for annotation purposes.

Format

```
$??ATS,tttt<CR><LF>
```

Format description

1 **??** = Talker identifier

2 **ATS** = telegram identifier

3 **tttt** = free text string

Proprietary third party telegrams and formats

All third party telegram formats supported by the EK60 are specified here. These telegram formats are created by third party organizations, and they are supported by the EK60 to allow for interface to third party systems.

Topics

Atlas depth telegram

This proprietary Atlas telegram contains the current depth from two channels.

Format

```
Dyxxxxx.xxm
```

Format description

1 **Dy** = Channel number; **DA** is channel number 1, **DB** is channel number 2.

2 **xxxxx.xx** = depth in meters

3 **m** = meters

Furuno GPhve heave telegram

This proprietary Furuno telegram format contains information about vessel heave.

Format

```
$PFEC,GPhve,xx.xxx,A*hh<CR><LF>
```

Format description

1 **$PFEC** = Talker

2 **GPhve** = Telegram identifier

3 **xx.xxx** = heave in meters and decimals

4 **A** = status

5 **hh** = checksum

File formats

This chapter describes the file formats supported by the EK60 Scientific echo sounder.

Topics

Numeric type definition

In order to describe the data type formats, common ”C” structures are used to represent individual data blocks.

Table 7 The size of the various “C” types

char	8-bit integer
WORD	16-bit unsigned integer
short	16-bit integer
Int	32-bit integer
long	32-bit integer
float	32-bit floating point (IEEE 754)
double	64-bit floating point (IEEE 754)
DWORDLONG	64-bit integer

Raw data format

The ***.raw** file may contain one or more of the following datagram types:

- Configuration
- NMEA
- Annotation
- Sample

Every ***.raw** file begins with a configuration telegram. A second configuration datagram within the file is illegal. The data content of the **Configuration** datagram of an already existing file cannot be altered from the EK60. **NMEA**, **Annotation** and **Sample** datagrams constitute the remaining file content. These datagrams are written to the ***.raw** file in the order that they are generated by the EK60.

In the following descriptions, the information after the "//" characters are comments to that line.

Note

Strictly sequential time tags are not guaranteed.

Topics

Data encapsulation

A standard encapsulation scheme is used for all data files. Each datagram is preceded by a 4 byte length tag stating the datagram length in bytes. An identical length tag is appended at the end of the datagram.

Format

```
long Length;
struct DatagramHeader
   {
   long DatagramType;
   struct {
      long LowDateTime;
      long HighDateTime;
   } DateTime;
   };
- -
< datagram content
- -
long Length;
};
```

Description

All datagrams use the same header. The datagram type field identifies the type of datagram. ASCII quadruples are used to ease human interpretation and long term maintenance; three characters identify the datagram type and one character identifies the version of the datagram.

The *DateTime* structure contains a 64-bit integer value stating the number of 100 nanosecond intervals since January 1, 1601. This is the internal "*filetime*" used by the Windows NT operating system. The data part of the datagram contains any number of bytes, and its content is highly datagram dependent.

Common computers fall into two categories:

- Intel based computers write a multibyte number to file starting with the LSB (Least Significant Byte).
- HP, Sun and Motorola do the opposite. They write the MSB (Most Significant Byte) to file first.

The byte order of the length tags and all binary fields within a datagram is always identical to the native byte order of the computer that writes the data file. It is the responsibility of the software that reads the file to perform byte swapping of all multibyte numbers within a datagram if required. Byte swapping is required whenever there is an apparent mismatch between the head and the tail length tags. Hence, the two length tags may be used to identify the byte order of the complete datagram.

The Intel processors allow a multibyte number to be located at any RAM address. However, this may be different on other processors; a short (2 byte) must be located at an even address, a long (4 byte) and a float (4 byte) must be located at addresses that can be divided by four. Hence, the numeric fields within a datagram is specified with this in mind.

Configuration datagram

All character strings are zero terminated.

Format

```
struct ConfigurationDatagram {
   DatagramHeader DgHeader // "CON0"
   ConfigurationHeader ConfigHeader;
   ConfigurationTransducer Transducer[];
   };
struct ConfigurationHeader
{
   char SurveyName[128]; // "Loch Ness"
   char TransectName[128];
   char SounderName[128]; // "ER60"
   char version [30];
   char spare [98];
   long TransducerCount; // 1 to 7
};
struct ConfigurationTransducer {
   char ChannelId[128]; // Channel identification
   long BeamType; // 0 = Single, 1 = Split
```

```
   float Frequency; // [Hz]
   float Gain; // [dB] - See note below
   float EquivalentBeamAngle; // [dB]
   float BeamWidthAlongship; // [degree]
   float BeamWidthAthwartship; // [degree]
   float AngleSensitivityAlongship;
   float AngleSensitivityAthwartship;
   float AngleOffsetAlongship; // [degree]
   float AngleOffsetAthwartship; // [degree]
   float PosX; // future use
   float PosY; // future use
   float PosZ; // future use
   float DirX; // future use
   float DirY; // future use
   float DirZ; // future use
   float PulseLengthTable[5];
      // Available pulse lengths for the channel [s]
   char Spare1[8]; // future use
   float GainTable[5];
      // Gain for each pulse length in the PulseLengthTable [dB]
   char Spare2[8]; // future use
   float SaCorrectionTable[5];
      // Sa correction for each pulse length in the PulseLengthTable [dB]
   char Spare3[8];
   char GPTSoftwareVersion [16];
   char Spare4[28];
   };
```

Note

float Gain: The single gain parameter was used actively in raw data files generated with software version 1.3. This was before *PulseLengthTable*, *GainTable* and *SaCorrectionTable* were introduced in software version 1.4 to enable gain and Sa correction parameters for each pulse duration.

NMEA datagram

This datagram contains the original NMEA 0183 input message line; carriage return, line feed and a terminating zero included.

Format

```
struct TextDatagram{
   DatagramHeader DgHeader; // "NME0"
   char Text[]; // "$GPGLL,5713.213,N......"
   };
```

Description and examples

The size of the datagram depends on the message length.

An example **GLL** NMEA position message line is shown below:

```
$GPGLL,5713.213,N,1041.458,E‹cr›‹lf›
```

The information contained in the **VTG** NMEA telegram is not used by the EK60, and this information is thus only written to the ***.raw** data file.

An example NMEA speed message line is shown below:

```
$HUVTG,245.0,T,245.0,M,4.0,N,2.2,K‹cr›‹lf›
```

Related topics

- *Specification of NMEA telegrams* on page 177

Annotation datagram

The annotation datagram contains comment text that you have entered ("dangerous wreck"). The text string is zero terminated. The size of the complete datagram depends on the annotation length. The maximum annotation string is 80 characters.

Format

```
struct TextDatagram {
   DatagramHeader DgHeader; // "TAG0"
   char Text[]; // "Dangerous wreck"
   };
```

Sample datagram

The sample datagram contains sample data from just one transducer channel. It can contain power sample data (Mode = 0), or it can contain both power and angle sample data (Mode = 1).

Format

```
struct SampleDatagram
   {
   DatagramHeaderDgHeader; // "RAW0"
   short Channel; // Channel number
   short Mode; // Datatype
   float TransducerDepth; // [m]
   float Frequency; // [Hz]
   float TransmitPower; // [W]
   float PulseLength; // [s]
   float BandWidth; // [Hz]
   float SampleInterval; // [s]
   float SoundVelocity; // [m/s]
   float AbsorptionCoefficient; // [dB/m]
   float Heave; // [m]
   float Tx Roll; // [deg]
   float Tx Pitch; // [deg]
   float Temperature; // [C]
   short Spare 1
   Short Spare 2
   float Rx Roll [Deg]
   float Rx Pitch [Deg]
   long Offset; // First sample
   long Count; // Number of samples
   short Power[]; // Compressed format - See Remark 1!
   short Angle[]; // See Remark 2 below!
   };
```

The sample data datagram can contain more than 32 768 sample points.

Remarks

1 **Power**: The power data contained in the sample datagram is compressed. In order to restore the correct value(s), you must decompress the value according to the equation below.

$$y = x \frac{10 \log 2}{256}$$

where:

- **x** = power value derived from the datagram
- **y** = converted value (in dB)

2 **Angle**: The fore-and-aft (alongship) and athwartship electrical angles are output as one 16-bit word. The alongship angle is the most significant byte while the athwartship angle is the least significant byte. Angle data is expressed in 2's complement format, and the resolution is given in steps of 180/128 electrical degrees per unit. Positive numbers denotes the fore and starboard directions.

Data subscription and remote control

The EK60 Scientific echo sounder system provides the facility to subscribe to data, and to control the system operation from a user developed remote application. This allows you to write your own application which controls the EK60 operation (e.g. start/stop pinging, change ping interval, and start/stop data recording).

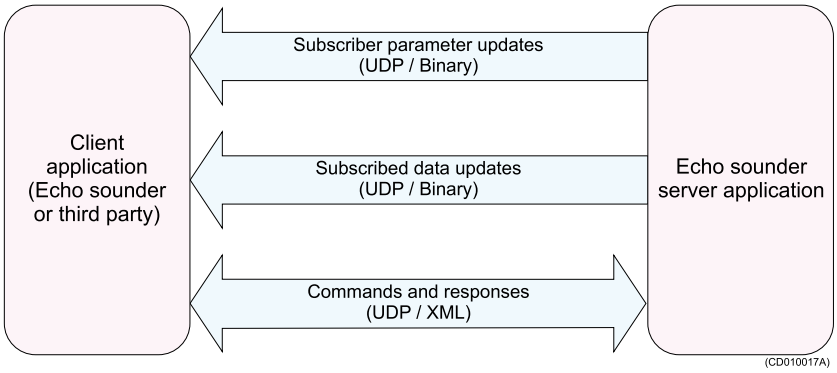
From your application you can also subscribe to data from the EK60. This means that you can ask the EK60 to continuously send various data (e.g. Depth data, Target Strength data, and Integration data) to your application.

Topics

Data subscription communication

The communication between the echo sounder program and your program is performed by exchanging UDP messages via the LAN. Command and response messages are XML based text messages. Subscribed data updates are binary data structures, which must be decoded using information about the data structure.

Figure 9 Data subscriptions



Data subscriptions overview

The following is an overview of the process for data subscription and remote control.

1 Request server information

2 Connect to server

3 Keep connection alive

4 Issue commands on server

- Subscription of data:
  - **a** Create data subscriptions
  - **b** Handle data
  - **c** Change data subscriptions
  - **d** Destroy data subscriptions
- Parameter management:
  - **a** Get parameter
  - **b** Set parameter
  - **c** Start parameter notifications
  - **d** Stop parameter notifications

5 Disconnecting from server

Request server information

Before you connect to the server running the echo sounder program, your client application must obtain information about the server's IP address and command port number.

RequestServerInfo

Send the following **RequestServerInfo** message to the specific IP address of the server, or broadcast the message to receive server information from all servers on the LAN. The message should be send to the UDP port number found in the Local port field on the **Server** page in the **Remoting** dialog in the server application.

```
struct RequestServerInfo
   {
   char Header[4]; // ”RSI\0”
   };
```

ServerInfo2

The server applications will respond to the requesting application with a message containing information about the server application. The content of the **ServerInfo2** message is shown below.

```
struct ServerInfo2
   {
   char Header[4]; // ”SI2\0”
   char ApplicationType[64];
   char ApplicationName[64];
         // Name of the current application
   char ApplicationDescription[128];
         // Description of the current application
   long ApplicationID;
         // ID of the current application
   long CommandPort;
         // Port number to send commands to
   long Mode;
         // If the application is
         // running against the local data source
         // or a remote data source
   char HostName[64];
         // IP address of the computer the
         // application is running on
   };
```

CommandPort

The server UDP port number **CommandPort** must be used from now on to send commands to the server application.

Connecting to server

Before commands can be sent to the server application, the client application must identify itself to the server application by sending a connect command, **ConnectRequest**, to the server. This connect command must contain user account and password information.

ConnectRequest

The contents of the connect message is shown below.

```
struct ConnectRequest
   {
   char Header[4]; // ”CON\0”
   char ClientInfo[1024];
      // e.g. ”Name:Simrad;Password:\0”
   };
```

Response

The server application will respond with a **Response** message containing at least the client id if the connect command succeeded, or an error if the connect command failed. The contents of the various parts of a response message for a successful connect command is shown below.

```
struct Response
   {
   char Header[4]; // ”RES\0”
   char Request[4]; // ”CON/0”
   char MsgControl[22]; // ”\0”
   char MsgResponse[1400];
         // Response text containing result of
         // connection request
   };
```

MsgResponse

The contents of the **MsgResponse** field consist of **ResultCode** and **Parameters**.

ResultCode: Result of the operation. The following values can be present:

- **S_OK**: operation was successful
- **E_ACCESSDENIED**: operation failed due to unknown account or wrong password
- **E_FAIL**: operation failed due to an unspecified error

Parameters: Comma separated list of various name:value pairs that may be present, only provided if the **ConnectRequest** is successful.

- **ClientID**: identification of the current client, to be used in all further communication with the server application
- **AccessLevel**: general access level for the current client

A successful connection will for example provide a **MsgResponse** containing:

```
ResultCode:S_OK,
Parameters:{ClientID:1,AccessLevel:1}\0
```

Connection failure

In case the connect command fails, the **ResponseField** will contain a **ResultInfo** field.

ResultInfo: Text describing the failure

Keep connection alive

Once the client application is connected to the server application a two-way monitoring of the application and communication ”health” must be started. This means that both the client and the server application must send an ”alive” message, **AliveReport**, periodically (each second).

AliveReport

The content of the **AliveReport** message is shown below.

```
struct AliveReport
   {
   char Header[4]; // ”ALI\0”
   char Info[1024]; // e.g. ”ClientID:1,SeqNo:1\0”
};
```

The **Info** field in an **AliveReport** message from the client to the server may contain (for example):

```
ClientID:1,SeqNo:1\0
```

SeqNo

The **SeqNo** part of the **Info** field shall contain the sequence number of the next request message from the client to the server (the sequence number shall start on 1). The server will use this to detect if any messages from the client have been lost, if a loss is detected, the server will issue a re-transmit request to the client. The **AliveReportDef** message from the server to the client may contain (for example):

```
SeqNo:1\0
```

Similar, the **SeqNo** part of the **Info** field shall contain the sequence number of the next response message from the server to the client. The client application can use this to detect if any message from the server is lost, and may then issue a re-transmit request to the server.

Issue commands on the server

Once connected, the following messages are used to issue commands.

Request

A **Request** message must be sent to the server application in order to issue a command on one of the available command targets in the server application. An example of the main elements of a command request is shown below.

```
struct Request
   {
   char Header[4]; // ”REQ\0”
   char MsgControl[22];
         // Sequence no, Current msg no, Total msg no
   char MsgRequest[1400];
         // XML based command request
};
```

MsgControl

The **MsgControl** field consists of the following parts:

- **Sequence number**: increments for each request message sent to the server.
- **Current message**: contains the current message in case a request must be split into several UDP messages.
- **Total message**: contains the total number of messages the current request consists of.

If the contents of **MsgControl** is ”2,1,3\0” it means that the current UDP message is the second request message sent from the client to the server, and that the current message is message number 1 of a request that consists of a total of three UDP messages.

MsgRequest

The contents of the **MsgRequest** depend on the current command target and will be described in detail later. For the moment it is enough to say that the format of the request is XML based and that the request must specify a command target, a method on the command target and any input parameters relevant for the current method. The general structure of a request is shown below.

```
<request>
   <clientInfo>
      <cid>clientid</cid>
      <rid>requestid</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>xx</targetComponent>
   <method>
   <yy>
      <zz></zz>
   </yy>
   </method>
</request>
```

where:

- **clientid** = client identification
- **requestid** = request identification
- **xx** = the name of the current command target
- **yy** = the name of the current method
- **zz** = any parameters of the current method

Response

The server application will respond with a **Response** message as shown below.

```
struct Response
   {
   char Header[4]; // ”RES\0”
   char Request[4]; // ”REQ/0”
   char MsgControl[22];
         // Sequence no, Current msg no, Total msg no
   char MsgResponse[1400];
         // XML based response text containing result
         // of command request
   };
```

The contents of the **Response** depend on the current command target and will be described in detail later. For the moment it is enough to say that the format of the response is XML based and that the response contains the result, any error messages, and any output parameters relevant for the current method. The general structure of a response is shown below.

```
<response>
   <clientInfo>
      <cid dt=”3”>clientid</cid>
      <rid dt=”3”>requestid</rid>
   </clientInfo>
   <fault>
      <detail>
      <errorcode dt=”3”>0</errorcode>
      </detail>
   </fault>
   <yyResponse>
      <zz dt=”3”></zz>
   </yyResponse>
</response>
```

where:

- **clientid** = client identification
- **requestid** = request identification
- **error code** = the result of the operation. 0 = OK
- **yy** = the name of the current method
- **zz** = any parameters of the current method

Collecting data

In order to collect echo sounder data from the server application, commands must be sent to the **RemoteDataServer** component of the server application. The following methods/commands are available.

- Create data subscription
- Handling data
- Change data subscription
- Destroy data subscription

For all commands a **Request** message must be sent from the client to the server - in the following sections only the contents of the **Request** field of the **Request** will be described.

For all commands a **Response** message will be sent from the server to the client - in the following sections only the contents of the **Response** field of the **Response** will be described.

Create data subscriptions

The method part of the request shall be set to **Subscribe**. The **Subscribe** method is defined with the following input parameters.

- **RequestedPort**: The local port on the client application that the data should be sent to
- **DataRequest**: The actual specification of the subscription

An example of the contents of the **Request** field of a **Subscribe** command to the **RemoteDataServer** is shown below:

```
<request>
   <clientInfo>
      <cid>1</cid>
      <rid>1</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>RemoteDataServer
   </targetComponent>
   <method>
      <Subscribe>
      <requestedPort>12345</requestedPort>
      <dataRequest>BottomDetection</dataRequest>
      </Subscribe>
   </method>
</request>
```

The server application will respond with a **Response** message. The **Subscribe** method has the following output parameters:

- **SubscriptionID**: The identification of the current subscription - can be used to differentiate between multiple subscriptions on the same port

An example of the contents of the **Response** field of a **Subscribe** command to the **RemoteDataServer** is shown below.

```
<response>
   <clientInfo>
      <cid dt="3">1</cid>
      <rid dt="3">1</rid>
   </clientInfo>
   <fault>
      <detail>
      <errorcode dt="3">0</errorcode>
      </detail>
   </fault>
   <SubscribeResponse>
      <subscriptionID dt="3"></subscriptionID >
   </SubscribeResponse>
</response>
```

Handling data

Data from the **RemoteDataServer** are wrapped in a **ProcessedData** structure.

```
struct ProcessedData
   {
   char Header[4]; // "PRD\0"
   long SeqNo;
         // Sequence number of the current
         // UDP message
   long SubscriptionID;
         // Identification of the current data
   unsigned short CurrentMsg;
         // Current message number
   unsigned short TotalMsg;
         // Total number of UDP messages
   unsigned short NoOfBytes;
         // Number of bytes in the following
         // Data field
   unsigned short Data[];
         // Actual data
};
```

If the amount of data exceeds the limit of one UDP message, the data will be split into multiple UDP messages. The **TotalMsg** field contains the number of UDP messages for the current data, this is identified with any number larger than 1. The **CurrentMsg** field contains the current message number out of the total number of messages.

The **Data** field shall be decoded according to the specified output structure from the subscription described in section *Data subscription types* on page 211.

Change data subscriptions

The method part of the request shall be set to **ChangeSubscription**. The **ChangeSubscription** method has the following parameters.

- **subscriptionID**: The id of the subscription that should be changed
- **dataRequest**: The actual specification of the subscription. This is described in the section *Data subscription types* on page 211.

An example of the contents of the **Request** field of a **ChangeSubscription** command to the **RemoteDataServer** is shown below.

```
<request>
   <clientInfo>
      <cid>1</cid>
      <rid>1</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>RemoteDataServer
      </targetComponent>
   <method>
      <ChangeSubscription>
         <subscriptionID>1</subscriptionID>
         <dataRequest>BottomDetection</dataRequest>
      </ChangeSubscription>
   </method>
</request>
```

The server application will respond with a **Response** message. The **ChangeSubscription** method does not have any output parameters.

An example of the contents of the **Response** field of a **ChangeSubscription** command to the **RemoteDataServer** is shown below.

```
<response>
   <clientInfo>
      <cid dt="3">1</cid>
      <rid dt="3">1</rid>
   </clientInfo>
   <fault>
      <detail>
         <errorcode dt="3">0</errorcode>
      </detail>
   </fault>
   <ChangeSubscriptionResponse>
      </ChangeSusbcriptionResponse>
</response>
```

Destroy data subscriptions

The method part of the request shall be set to **Unsubscribe**. The **Unsubscribe** method has the following parameters.

- **subscriptionID**: The identification of the subscription that shall be closed

An example of the contents of the **szRequest** field of an **Unsubscribe** command to the **RemoteDataServer** is shown below.

```
<request>
   <clientInfo>
      <cid>1</cid>
      <rid>1</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>RemoteDataServer
      </targetComponent>
   <method>
      <Unsubscribe>
      <subscriptionID>1</subscriptionID>
      </Unsubscribe>
   </method>
</request>
```

The server application will respond with a **ResponseDef** message. The **Unsubscribe** method does not have any output parameters.

An example of the contents of the **Response** field of an **Unsubscribe** command to the **RemoteDataServer** is shown below.

```
<response>
   <clientInfo>
      <cid dt="3">1</cid>
      <rid dt="3">1</rid>
   </clientInfo>
   <fault>
      <detail>
         <errorcode dt="3">0</errorcode>
      </detail>
   </fault>
   <UnsubscribeResponse>
      </UnsubscribeResponse>
</response>
```

Parameter management

In order to set/get parameters in the server application, commands must be sent to the **ParameterServer** component of the server application. The following methods/commands are available:

- Get parameter value/attribute
- Set parameter value/attribute
- Subscribe on parameter value/attribute change notifications
- Unsubscribe parameter notifications

Get parameter value/attribute

The method part of the request shall be set to **GetParameter**. The server application will respond with a **Response** message.

Input parameters

The **GetParameter** method has the following input parameters:

- **ParamName**: Full name of the current parameter
- **Time**: The time where the value should be read (only available for some parameters). Use 0 if latest value is wanted.

An example of the contents of the **Request** field of a **GetParameter** command to the **ParameterServer** is shown below.

```
<request>
   <clientInfo>
      <cid>1</cid>
      <rid>28</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>
      ParameterServer
   </targetComponent>
   <method>
      <GetParameter>
         <paramName>
         RemoteCommandDispatcher/ClientTimeoutLimit
         </paramName>
         <time>0</time>
      </GetParameter>
   </method>
</request>
```

Output parameters

The **GetParameter** method has the following output parameters:

- **Value**: The value of the parameter
- **Time**: The time when the parameter was updated

An example of the contents of the **Response** field of a **GetParameter** command to the **ParameterServer** is shown below.

```
<response>
   <clientInfo>
      <cid dt="3">1</cid>
      <rid dt="3">28</rid>
   </clientInfo>
   <fault>
      <detail>
         <errorcode dt="3">0</errorcode>
      </detail>
   </fault>
   <GetParameterResponse>
      <paramValue>
         <value>60</value>
         <time>0</time>
      <paramValue/>
   <GetParameterResponse/>
</response>
```

Set parameter value/attribute

The method part of the request shall be set to **SetParameter**. The server application will respond with a **Response** message.

Input parameters

The **SetParameter** method has the following input parameters:

- **ParamName**: Full name of the current parameter
- **paramValue**: The new value to update the parameter with
- **paramType**: The data type of the **paramValue** field

An example of the contents of the **Request** field of a **SetParameter** command to the **ParameterServer** is shown below.

```
<request>
   <clientInfo>
      <cid>1</cid>
      <rid>28</rid>
   </clientInfo>
   <type>invokeMethod</type>
   <targetComponent>
      ParameterServer
   </targetComponent>
   <method>
      <SetParameter>
         <paramName>
         RemoteCommandDispatcher/ClientTimeoutLimit
         </paramName>
         <paramValue>60</paramValue>
         <paramType>3</paramType>
      </SetParameter>
   </method>
</request>
```

Output parameters

The **SetParameter** method does not have any output parameters.

An example of the contents of the **Response** field of a **SetParameter** command to the **ParameterServer** is shown below.

```
<response>
   <clientInfo>
      <cid dt=”3”>1</cid>
      <rid dt=”3”>28</rid>
   </clientInfo>
   <fault>
      <detail>
      <errorcode dt=”3”>0</errorcode>
      </detail>
   </fault>
</response>
```

Disconnecting from server

The client application shall send a **DisconnectRequestDef** message to the server application when the client is finished with its operation against the server.

The parameters are:

- **Header**: DIS\0
- **szClientInfo**: Name:Simrad;Password:\0

Data subscription types

This section describes the available data subscriptions in the echo sounder.

All data subscriptions require the **ChannelID** (channel identifier) for the frequency channel from which the subscription data is requested. A comma separated list of available identifiers can be obtained using the **ParameterServer** component to get the parameter **TransceiverMgr** Channels.

The data output will always start with a 64-bit integer time value, which identifies the number of 100 nanoseconds intervals that has elapsed since January 1, 1601. If a parameter is skipped in the subscription input string it will be replaced by the default value. The decoding of the subscription will be case insensitive. In the output data to be received by external devices, the information about the current structure length is given. From this information, the number of elements in an array may be calculated.

Bottom detection

Subscription type string: **BottomDetection**.

Input

Parameters	Range	Default	Unit
UpperDetectorLimit	(0,20000)	0	m
LowerDetectorLimit	(0,20000)	1000	m
BottomBackstep	(-200,100)	–50	dB

Output

```
struct StructBottomDepthHeader
   {
      DWORDLONG dlTime;
   };
struct StructBottomDepthData
   {
      double dBottomDepth;
         // detected bottom depth [meter]
      double dReflectivity;
         // bottom surface backscatter [dB]
      double dVesselLogDistance;
         // sailed distance [nmi]
   };
struct StructBottomDepth
   {
      StructBottomDepthHeader BottomDepthHeader;
      StructBottomDepthData BottomDepthData;
   };
```

Example

Building a **BottomDetection** subscription string:

```
BottomDetection, ChannelID=<ChannelID>,
UpperDetectorLimit=3.0, LowerDetectorLimit=500.0,
BottomBackstep=-60.0
```

Target strength (TS) detection

Subscription type string: **TSDetection**.

Input

Parameters	Range	Default	Unit
sLayerType	(Surface. Bottom, Pelagic)	Surface	None
Range	(0,20000)	10000	m

Parameters	Range	Default	Unit
RangeStart	(0,20000)	0	m
MinTSValue	(–120,50)	–50.0	dB
MinEchoLength	(0,20)	0.8	None
MaxEchoLength	(0,20)	1.8	None
MaxGainCompensation	(0,12)	6.0	None
MaxPhaseDeviation	(0,100)	8.0	Phase steps

Output

```
struct StructTSDataHeader
   {
   DWORDLONG dlTime;
   };
struct StructEchoTrace
   {
   double Depth;
      // Target depth [meter]
   double TSComp;
      // Compendated TS [dB]
   double TSUncomp;
      // Uncompensated TS [dB]
   double AlongshipAngle;
      // Alongship angle [deg]
   double AthwartshipAngle;
      // Athwartship angle [deg]
   double sa;
      // Sa value for target
   };
struct StructTSDataBody
   {
   WORD NoOfEchoTraces;
      // Number of targets accepted in ping
   StructEchoTrace EchoTraceElement[100];
   };
struct StructTSData
   {
   StructTSDataHeader TSDataHeader;
   StructTSDataBody TSDataBody;
   };
```

Example

Building a **TSDetection** subscription string:

```
TSDetection, ChannelID=<ChannelID>,
LayerType=Surface, Range=200,
RangeStart=3, MinTSValue=-55,
MinEcholength=0.7, MaxEcholength=2.0,
MaxGainCompensation=6.0, MaxPhasedeviation=7.0
```

Sample data

Subscription type string: **SampleData**.

Input

Parameters	Range	Default	Unit
SampleDataType	(Power, Angle, Sv, Sp, Ss, TVG20, TBG40, PowerAngle)	Power	None
Range	(0,20000)	100	m
RangeStart	(0,20000)	0	m

Output for Power, Angle, Sv, Sp, Ss, TVG20 and TVG40

```
struct StructSampleDataHeader
   {
   DWORDLONG dlTime;
   };
struct StructSampleDataArray
   {
   short nSampleDataElement[30000];
      // 16-bits sample in logarithmic format
   };
struct StructSampleData
   {
   StructSampleDataHeader SampleDataHeader;
   StructSampleDataArray SampleDataArray;
   };
```

Output for PowerAngle

```
struct StructSampleDataHeader
   {
   DWORDLONG dlTime;
   };
struct StructSamplePowerAngleArray
   {
   short nSampleDataElement[60000];
      // Composite sample array for power
      // and angle
   };
struct StructSamplePowerAngleValues
   {
   int nPowerValues;
      // Number of power samples
   int nAngleValues;
      // Number of angle values
   };
struct StructSamplePowerAngle
   {
   StructSampleDataHeader SampleDataHeader;
   StructSamplePowerAngleValues SamplePowerAngleValues;
   StructSamplePowerAngleArray SamplePowerAngleArray;
   };
```

Example

Building an **SampleData** subscription string:

```
SampleData,ChannelID=<ChannelID>,
SampleDataType=Power, Range=100,
RangeStart=10
```

Echogram

Subscription type string: **Echogram**.

Input

Parameters	Range	Default	Unit
PixelCount	(0,10000)	500	None
Range	(0,20000)	100	m
RangeStart	(0,20000)	0	m
TVGType	(Pr, Sv, Sp, TS, SpAndTS)	Sv	None
EchogramType	(Surface, Bottom, Trawl)	Surface	None
CompressionType	(Mean, Peak)	Mean	None
ExpansionType	(Interpolation, Copy)	Interpolation	None

Output

```
struct StructEchogramHeader
   {
      DWORDLONG dlTime;
   };
   struct StructEchogramArray
   {
      short nEchogramElement[30000];
         // 16-bit logarithmic format
   };
struct StructEchogram
   {
      StructEchogramHeader EchogramHeader;
      StructEchogramArray EchogramArray;
};
```

Example

Building a **Echogram** subscription string:

```
Echogram, ChannelID=<ChannelID>,
PixelCount=500, Range=100, RangeStart=0,
TVGType=TS, EchogramType=Surface,
CompressionType=Mean, ExpansionType=Interpolation
```

Targets echogram

Subscription type string: **TargetsEchogram** .

This subscription will only produce an echogram array containing detected echo traces with their compensated TS values between the transmit pulse and the bottom. Below bottom the selected TVG type is used.

Input

Parameters	Range	Default	Unit
PixelCount	(0,10000)	500	None
Range	(0,20000)	100	m
RangeStart	(0,20000)	0	m
TVGType	(TS, SP, Ts)	Must be set	None
EchogramType	(Surface, Bottom)	Surface	None

Parameters	Range	Default	Unit
MinTSValue	(-120,50)	–50	dB
MinEchoLength	(0,20)	0.8	None
MaxEchoLength	(0,20)	1.8	None
MaxGainCompensation	(0,12)	6.0	dB
MaxPhaseDeviation	(0,100)	8.0	Phase steps

Output

```
struct StructEchogramHeader
   {
      DWORDLONG dlTime;
   };
struct StructEchogramArray
   {
      short nEchogramElement[30000];
   };
struct StructEchogram
   {
      StructEchogramHeader EchogramHeader;
      StructEchogramArray EchogramArray;
};
```

Example

Building a **TargetsEchogram** subscription string:

```
TargetsEchogram, ChannelID=<ChannelID>,
PixelCount=500, Range=100, RangeStart=0,
TVGType=TS, EchogramType=Surface, MinTSValue=-55.0,
MinEcholength=0.7, MaxEcholength=2.0,
MaxGainCompensation=6.0, MaxPhasedeviation=7.0
```

Integration

Subscription type string: **Integration**.

The update of Sa will be enabled by setting the **Integration State** to start. If the update parameter is set to **Update Ping**, the new Sa value is received for every ping. If the **Update Accumulate** is set, the Sa is received only when the **Integration State** changes to Stop.

Input

Parameters	Range	Default	Unit
LayerType	(Surface, Bottom, Pelagic)	Surface	None
IntegrationState	(Start, Stop)	Start	None
Update	(UpdatePing, UpdateAccumulate)	UpdatePing	None
Range	(0,20000)	100	m
RangeStart	(0,20000)	10	m
Margin	(0,200)	1	m
SvThreshold	(–200,100)	–100	dB

Output

```
struct StructIntegrationDataHeader
   {
      DWORDLONG dlTime;
   };
struct StructIntegrationDataBody
   {
      double dSa;
         // integrated value [m2/nmi2]
   };
struct StructIntegrationData
   {
      StructIntegrationDataHeader IntegrationDataHeader;
      StructIntegrationDataBody IntegrationDataBody;
};
```

Example

Building a **Integration** subscription string:

```
Integration, ChannelID=<ChannelID>,
State=Start, Update=UpdatePing,
Layertype=Surface, Range=100,
Rangestart=10, Margin=0.5, SvThreshold=-100.0
```

Targets integration

Subscription type string: **TargetsIntegration**.

This is a composite subscription where TS detection and integration parameters must be set. The Sa value is taken only from the accepted single echo trace inside the range.

Input

Parameters	Range	Default	Unit
sLayerType	(Surface, Bottom, Pelagic)	Surface	None
sIntegrationState	(Start, Stop)	Start	None
Update	(UpdatePing, UpdateAccumulate)	UpdatePing	None
Range	(0,20000)	100	m
RangeStart	(0,20000)	10	m
Margin	(0,200)	1	m
SvThreshold	(–200,100)	–100	dB
MinTSValue	(–120,50)	–50	dB
MinEchoLength	(0,20)	0.8	None
MaxEchoLength	(0,20)	1.8	None
MaxGainCompensation	(0,12)	6.0	dB
MaxPhaseDeviation	(0,100)	8.0	Phase steps

Output

```
struct StructIntegrationDataHeader
   {
```

```
      DWORDLONG dlTime;
   };
struct StructIntegrationDataBody
   {
      double dSa;
         // integrated value from single echo
         // trace [m2/nmi2]
   };
struct StructIntegrationData
   {
      StructIntegrationDataHeader IntegrationDataHeader;
      StructIntegrationDataBody IntegrationDataBody;
   };
```

Example

Building a **TargetsIntegration** subscription string:

```
TargetsIntegration, ChannelID=<ChannelID>,
State=Start, Layertype=Surface, Range=100,
Rangestart=10, Margin=0.5, SvThreshold=-100.0,
MinTSValue=-55.0, MinEcholength=0.7,
MaxEcholength=2.0, MaxGainCompensation=6.0,
MaxPhasedeviation=7.0
```

Parameter descriptions

The **ParameterServer** component can be used to access asynchronous and ping based parameters. The parameters can be “read”, “set” or subscribed to. A subscription will notify only when the parameter’s value is changed.

The following is a list of the most relevant parameters. The parameter name must be used when working with parameters as described in section *Parameter management* on page 209.

Parameters identified as “read only” (R/O) can not be set. Some of them, for example sensor parameters, can be set, but the new value will immediately be over-written if a live sensor is connected.

Table 8 Ping based parameters

Description	Parameter name	R/O	Range	Unit
List of ChannelID’s	TransceiverMgr/ Channels	Yes	N/A	-
Frequency	TransceiverMgr/ <ChannelID>/ Frequency	Yes	1.000 to 1.000.000	Hz
Pulse length	TransceiverMgr/ <ChannelID>/ PulseLength	No	0,000 to 0,065535	sec
Sample interval	TransceiverMgr/ <ChannelID>/ SampleInterval	No	0,000010 to 0,065535	sec
Transmit power	TransceiverMgr/ <ChannelID>/ TransmitPower	No	0 to 10.000	W

Table 8 Ping based parameters (cont'd.)

Description	Parameter name	R/O	Range	Unit
Absorption Coefficient	TransceiverMgr/ <ChannelID>/ AbsorptionCofficient	Yes	0,0 to 0,3	dB/m
Sound velocity	TransceiverMgr/ <ChannelID>/ SoundVelocity	No	1.400 to 1.700	m/s
Transducer name	TransceiverMgr/ <ChannelID>/ TransducerName	Yes	N/A	-
Transducer depth	TransceiverMgr/ <ChannelID>/ TransducerDepth	No	0 to 10.000	m
Equivalent beam angle	TransceiverMgr/ <ChannelID>/ EquivalentBeamAngle	No	–100 to 0	dB
Angle sensitivity alongship	TransceiverMgr/ <ChannelID>/ AngleSensitivityAlongship	No	0 to 100	el.deg/mec.deg
Angle sensitivity athwartship	TransceiverMgr/ <ChannelID>/ AngleSensitivityAthwartship	No	0 to 100	el.deg/mec.deg
Beamwidth alongship	TransceiverMgr/ <ChannelID>/ BeamWidthAlongship	No	0 to 100	deg
Beamwidth athwartship	TransceiverMgr/ <ChannelID>/ BeamWidthAthwartship	No	0 to 100	deg
Angle offset alongship	TransceiverMgr/ <ChannelID>/ AngleOffsetAlongship	No	–10 to 10	deg
Angle offset athwartship	TransceiverMgr/ <ChannelID>/ AngleOffsetAthwartship	No	–10 to 10	deg
Gain	TransceiverMgr/ <ChannelID>/ Gain	No	1 to 100	dB
Sa correction	TransceiverMgr/ <ChannelID>/ SaCorrection	No	–10 to 10	dB
Ping time	TransceiverMgr/ PingTime	Yes	2E64	100 ns step
Vessel latitude	TransceiverMgr/ Latitude	No	–90 to 90	deg
Vessel longitude	TransceiverMgr/ Longitude	No	–180 to 180	deg
Vessel heave	TransceiverMgr/ Heave	No	–100 to 100	m

Table 8 Ping based parameters (cont'd.)

Description	Parameter name	R/O	Range	Unit
Vessel roll	TransceiverMgr/ Roll	No	–90 to 90	deg
Vessel pitch	TransceiverMgr/ Pitch	No	–90 to 90	deg
Vessel distance	TransceiverMgr/ Distance	Yes	0 to 100.000	nmi
Noise estimate	ProcessingMgr/ <ChannelID>/ ChannelProcessingCommon/ NoiseEstimate	Yes	0 to –200	dB

Table 9 Asynchronous parameters

Description	Parameter name	R/O	Range	Unit
Vessel speed	OwnShip/ Speed	No	0 to 100	m/s
Vessel latitude	OwnShip/ Latitude	No	–90 to 90	deg
Vessel longitude	OwnShip/ Longitude	No	–180 to 180	deg
Vessel heave	OwnShip/ Heave	No	–100 to 100	m
Vessel roll	OwnShip/ Roll	No	–90 to 90	deg
Vessel pitch	OwnShip/ Pitch	No	–90 to 90	deg
Vessel distance	OwnShip/ VesselDistance	No	0 to 100.000	nmi
Environment temperature	OwnShip/ EnvironmentData/ Temperature	No	–5 to 50	deg
Environment salinity	OwnShip/ EnvironmentData/ Salinity	No	0 to 0,01	–
Environment sound velocity	OwnShip/ EnvironmentData/ SoundVelocity	No	1400 to 1700	m/s

Table 10 Operation mode parameters

Description	Parameter name	R/O	Range	Unit
Ping start/stop	OperationControl/ OperationMode	No	Last bit: 0 = Stop, 1 = Start	
Ping rate mode	AcousticDeviceSynchroniser/ SyncMode	No	1 = Interval, 2 = Maximum, 32 = Single step	

Table 10 Operation mode parameters (cont'd.)

Description	Parameter name	R/O	Range	Unit
Ping interval	AcousticDeviceSynchroniser/ Interval	No	Larger than 10	msec
Save raw data on/off	AcousticDeviceSynchroniser/ SaveRawData	No	0 = Off, 1 = On	

Echo sounder theory

When you use an echo sounder there are some basic knowledge that you may find it useful to possess.

Topics

Concepts

Observe the following descriptions of key concepts.

Topics

Observation range

Absorption increases dramatically with frequency in salt water. For maximum observation range you should select a low operating frequency, a large transducer and the maximum transmit power.

Typical observation ranges are shown in the table. Using the **Simrad ES38B** transducer (38 kHz, 7x7 degrees, 2000 W) you can observe a 60 centimeter cod down to 950 meters, and bottom detection works down to 2800 meters. However, with the **Simrad ES200–7C** transducer (200 kHz, 7x7 degrees, 1000 W) you can only observe that same cod down to 270 meters, and bottom detection becomes unreliable below 500 meters.

Table 11 Maximum detection depth, single beam transducers

Transducer	Frequency (kHz)	Pulse duration (ms)	Bandwidth (hz)	Tx power (W)	Range fish (m)	Range bottom (m)
12–16	12	16,4	193	2000	850	10000
27–26	27	8,18	387	3000	1100	4400
38/200D	38	4,09	766	1000	500	2100
38–9	38	4,09	766	1500	800	2600
38–7	38	4,09	766	2000	950	2800
50/200D	50	2,05	1493	1000	500	1500
50–7	50	2,05	1493	2000	700	1900
120–25	120	1,02	3026	1000	390	800
50/200D	200	1,02	3088	1000	280	550

Table 12 Maximum detection depth, split beam transducers

Transducer	Frequency (kHz)	Pulse duration (ms)	Bandwidth (hz)	Tx power (W)	Range fish (m)	Range bottom (m)
ES18–11	18	8,21	382	2000	1100	7000
ES38B	38	4,09	766	2000	950	2800
ES70–11	70	2,05	1526	800	450	1100
ES120–7C	120	1,02	3026	1000	440	850
ES200–7C	200	1,02	3088	1000	270	550

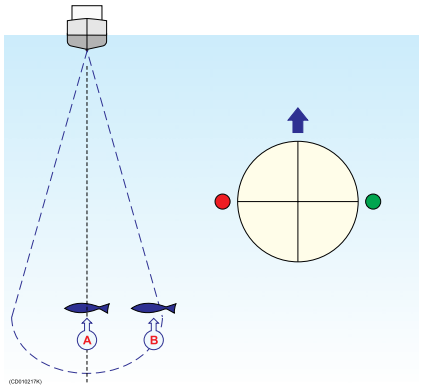
These range calculations assume a normal sea water salinity (3.5 ppt) and temperature (+10°C), an average bottom (surface backscattering strength = -20 dB) and a noise level typical for a moving vessel.

Split-beam operation

The EK60 uses the split-beam technique for assessment of the size distribution of individual fish. A split-beam transducer is electrically divided into four quadrants. All four quadrants are excited in parallel during transmission. However, the received signal from each quadrant is separately amplified in a four-channel matched receiver allowing the direction of arrival of an echo to be determined.

An acoustic wave front propagating towards the transducer arrives at the four quadrants at different times causing the phase angle of the electrical output signal from the quadrants to differ. The fore-and-aft angle is determined from the electrical phase difference between the fore and the aft transducer halves, and the athwartships angle is determined from the starboard and port signals.

Figure 10 Split beam principles



Fish A is positioned along the transducer axis where the transducer has its maximum sensitivity, while **Fish B** is positioned towards the edge of the beam where the sensitivity is lower. Evidently, the echo signal from **Fish A** will be stronger than the signal from **Fish B** even though they are of the same size and at the same depth. Hence, determining fish size from the received echo strength alone will not be too successful. A split-beam echo sounder measures the position of the fish within the beam. The sounder corrects for the difference in transducer sensitivity and computes the true size of the fish.

The split-beam measurement technique only works for echoes originating from one single fish since the electrical phase will be random if echoes from multiple individuals at different positions in the beam are received simultaneously.

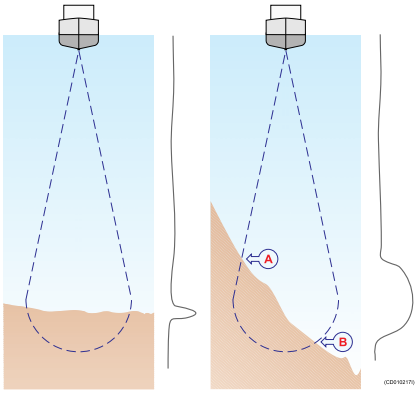
Consequently, measurement of fish size inside a school of fish tends to be unreliable.

Bottom echo

A hard flat bottom reflects the transmitted signal as if it were a mirror. The transmitted pulse hits the illuminated bottom area at nearly the same instant, and the echo from different parts of this area arrive back at the surface also at nearly the same instant.

The received echo signal is basically an attenuated copy of the short transmit pulse. The echo signal from a sloped bottom is characterized by having a longer duration and a slower rise and fall time. The transmitted pulse first hits the slope at point (**A**), and as time elapses the reflection point travels along the slope towards point (**B**). Many locations do not have a solid hard bottom. Frequently, the bottom is composed of layers of mud, clay and sand which can be observed as coloured bands on the echo sounder display.

Figure 11 Bottom echo principles



The bottom detection algorithm is implemented solely in software, and separate algorithms are run for each frequency channel. The algorithm is designed with emphasis on reliability in the sense that erroneous depth detections are never output. Whenever the quality of a detection is questionable the algorithm outputs a depth of 0.00 to indicate that no reliable detection was obtained. The EK60 algorithm is designed to handle a number of difficult situations. The algorithm maintains bottom lock for a discontinuous jump in bottom depth. It avoids false bottom detections on a dense school of fish. The algorithm chooses the upper boundary of the first layer when the bottom consists of layers.

The bottom detection algorithm locks to the first good bottom return. The depth at point A rather than the depth along the transducer axis will be output for a sloped bottom. The detected depth value is always smaller than the depth along the transducer axis implying that a safety margin is automatically included.

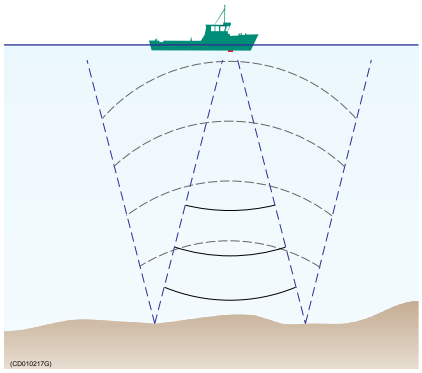
Wave propagation

The velocity of sound wave propagation in the sea varies slightly with temperature, salinity and pressure. The velocity varies between 1440 and 1520 m/s in shallow sea water, while a velocity around 1480 m/s can be expected at 1000 m depth. In shallow fresh water the velocity is approximately 1430 m/s.

A good average value to be used in the **Environment** dialog is 1470 m/s.

The EK60 transmits high energy sound wave pulses into the sea. A flat bottom reflects the transmitted wave as if it were a mirror. The propagating energy is spread over a larger and larger area as it travels down to the bottom and up again. The energy is spread over a four times larger area every time the travel distance doubles.

Figure 12 Wave propagation from a flat bottom

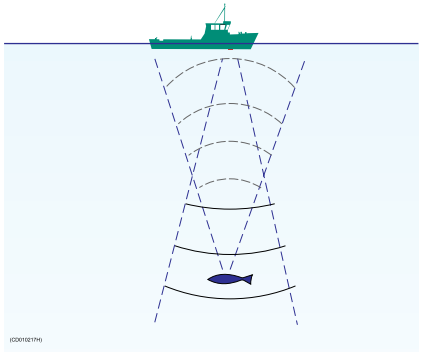


A large school of fish reflects sound waves similarly. This type of spreading is referred to as *square-law* or *20 log TVG (Time Varying Gain)* spreading.

The situation is slightly different when observing the echoes from individual fish. The transmitted wave undergoes square-law spreading when travelling from the surface and down to the fish. The swim bladder of the fish scatters a small fraction of the arriving energy in all directions. Travelling from the fish and back towards the surface the scattered wave undergoes another square-law spreading. The combined effect is referred to as *quad-law* or *40 log TVG* spreading.

In the echo sounder's **Echogram** dialog 20 log TVG spreading is referred to as *School Gain* and *Bottom Gain*, while 40 log TVG spreading is referred to as *Fish Gain*.

Figure 13 Wave propagation from a fish



Propagation losses due to absorption are much higher in sea water than in fresh water. Absorption also increases with frequency. At 38 kHz the absorption is 0.5 dB/km in fresh water and 10 dB/km in sea water. At 200 kHz the absorption is 10 dB/km in fresh water and 50 dB/km in salt water. The echo sounder must know which water type is present in order to compensate for these losses correctly.

The dB (decibel) unit has long traditions in underwater acoustics and other fields in physics. It is a logarithmic measure for the ratio between two quantities.

Biomass

Provided that you use an EK60 with a split beam transducer, the numerical view will contain a value for biomass.

This biomass value is an indicator to how much fish you currently have in the current echogram. Every single fish will emit an echo, and the sum of all these registered echoes are presented as a number. Smaller organisms such as plankton will also emit echoes, but these are so weak that they will hardly influence on the total biomass.

The EK60 records all the targets from the smallest plankton to the largest whale, and provides these findings as a number. For all practical purposes this number will provide you with information about the fish abundance to allow you to decide if it pays off

to start fishing. You must also consider if this number is a result of large amounts of plankton or bait, or if there is real fish below the keel. The number provided to display fish abundance is relative, and after some use your experience will be a valuable factor when the decision is made.

The biomass value is also used by the researches to calculate how much fish there is in the ocean. If you know the fish specie and the size of the fish, you can calculate number of individual fishes for a given volume of sea. Other means to establish the final result are trawling and catch data from the fishery community.

Note

If you have other echo sounders or sonars running asynchronous with the EK60, the EK60 will also measure the transmit pulse from the secondary system. This is called interference.

A full synchronization of the various acoustic instruments is required. If your own vessel produces excessive noise this will also be taken into the biomass calculations and provide you with inaccurate information.

Related topics

- *Echogram* on page 144

Dynamic range and display presentation

The EK60 echo sounder has a dynamic range of 140 dB. This means that the sounder can receive both very strong and very weak echoes. Actually, the EK60 will detect echoes from plankton to whales, bottom on most depths, and present the information free from distortion. As a comparison, our old echo sounders ES380 and ET100 had - using analogue TVG - a dynamic range corresponding to approximately 65 dB.

Naturally, we can not present all these echoes on the display simultaneously, as this would create a mess of colours.

When 12 colours are used, we create a 36 dB section and give each colour a 3 dB strength. Every colour (3 dB) the represent a doubling of the echo strength. With 12 colours in use this will be a 36 dB colour range from grey to brown. Grey is used for the weakest echoes, while the strongest echoes are brown. All echoes stronger than brown will still be brown, while echoes weaker than grey will not be shown.

With 64 colours in use, each colour represents approximately 0,5 dB echo strength.

The old paper sounders had a dynamic range of 12 dB in their printouts using the “colours” from light grey to black. The dynamic range in the EK60 colour presentation is thus a lot larger; 24 dB or 250 times.

Bottom slopes

“Bottom slopes” is a well known phenomenon with echo sounders. This happens when the bottom rises suddenly, and the start edge of the transducer beam detects the bottom before the opposite edge.

One method for minimizing the “bottom slopes” phenomenon is to use a transducer with narrow beam, or to increase the pulse duration.

To minimize this phenomenon on the EK60 you can open the **Bottom Detector** dialog by right-clicking the depth presentation, and then change the setting for **Backstep Minimum Level**.

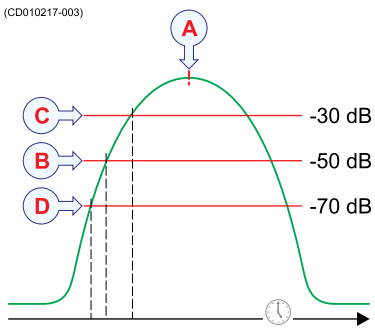
A *The peak of the bottom pulse*

B *Default bottom backstep level*

C *Approximate bottom backstep level for flatfish detection*

D *Approximate bottom backstep level for seagrass detection*

Figure 14 The Bottom Backstep principle



The bottom pulse basically identifies the bottom depth just prior to the peak of the pulse (A). However, this may not be the true bottom. For example, if the bottom pulse is generated by a rock bottom under a thick layer of mud, the actual depth is slightly shallower. For this reason, the EK60 is by default set up to give you a depth reading a few milliseconds before the peak of the pulse. This is done by setting the bottom backstep level to a default value of -50 dB (B).

By further decreasing the level (make it more negative) the bottom detector will become more “sensitive”, and the bottom will be detected earlier.

On the echogram the white line will then “climb” up the slope. Make sure that you do not increase the sensitivity too much. This will have an effect on the fish detection on a flat bottom, and the biomass values will be wrong. Our experience show that an approximately -75 dB bottom backstep value can be used safely.

By increasing the bottom backstep value (make it more positive) the bottom will be detected later, and it will appear to be deeper.

Tip

Rule of thumb:

- Reduce the bottom backstep level (make it more negative) to detect the bottom earlier, and thus increase bottom detection “sensitivity”.
- Increase the bottom backstep level (make it more positive) to detect the bottom later, and thus increase “penetration”.

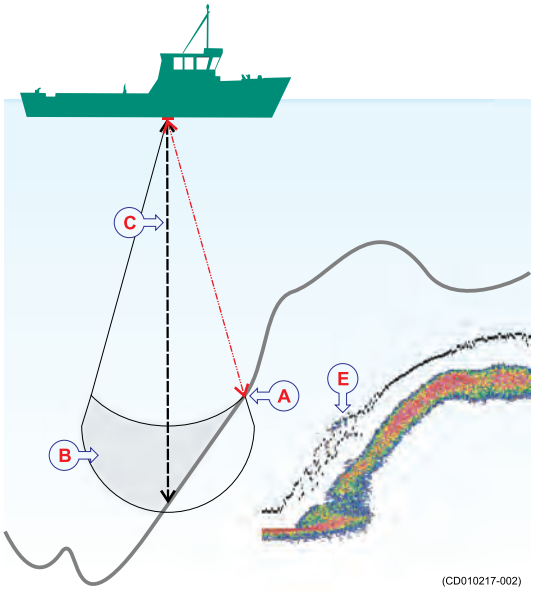
Example 6 Bottom slopes

The edge of the beam (A) hits the bottom first, and starts to give an echo. The bottom detector in the EK60 measures the strongest echo, detects what it thinks is the bottom, and starts the white line.

The area above the bottom detection (B) will be masked off, and even though it may contain fish these will not be visible because the echo from the bottom is stronger than those from the fish. The estimated depth (C) will be shown.

On the EK60 the phenomenon will cause the bottom line (E) to be distorted.

Figure 15 Bottom slopes



Parameters

Observe the following descriptions of key parameters.

Topics

TVG gain

TVG means *Time Varied Gain*. When TVG is used in an echo sounder, we also some times refer to it as *depth variable gain*.

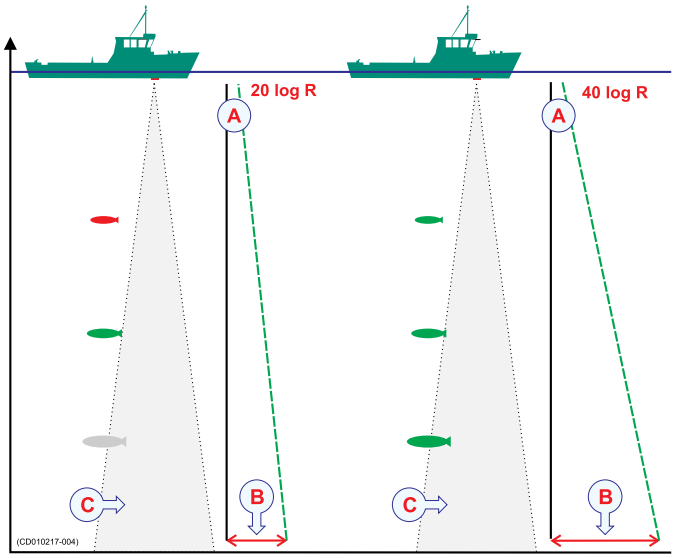
The purpose of the TVG functionality is to make all fishes appear with the same echo colour independent of their different depths.

In more technical terms, time varied gain (TVG) is a signal compensation. When the acoustic signal is transmitted from the echo sounder transducer, it is subjected to loss due to *absorption* and *spreading*. First, depending on the current salinity and temperature, the water will absorb some of the energy from the transmission. The absorption loss increases as the range increases. Second, the energy will spread out to form a circular beam. The width of this beam also increases with the range. Both absorption and spreading will thus reduce the energy, and both will also have an effect on the returned echo signal. The TVG compensation is designed to counteract these natural phenomena, and this is done in the EK60 using digital signal processing. The desired result is that fish of the same size return echoes of the same strength (colour), regardless of range.

Because the strength of the echoes will become weaker with increasing depth, the echo sounder will automatically amplify the deepest echoes more than the shallower echoes. In fact, the gain will increase proportional to how long the echo sounder “waits” for the echoes. When you choose the TVG setting you can either switch it off (which we do NOT recommended), or you can choose settings **20 log R** or **40 log R**.

The various settings control the gain algorithms, how much gain to be applied when the depth increases. When you choose the **40 log R** setting, the gain will increase with the depth more rapidly than if you choose the **20 log R** setting. This is simply because individual fishes emit smaller echoes than a school, and this makes them more difficult to detect. In the equation the character **R** means “Range”.

Figure 16 TVG principle



The illustration shows how the gain close to the bottom (B) is larger than just below the transducer (A). The echoes from the fish close to the bottom will then be shown with the same strength (colour) as echoes from pelagic fish.

The left vessel uses the **20 log R** setting. Due to the increasing beam width (C), single fishes are shown larger and larger with increasing depth, even though they may be of identical size.

The right vessel uses the **40 log R** setting. The size of the fish will still appear to grow larger as the range increases, but the echo is compensated differently to offer a more uniform echo strength (colour in the echogram).

When you are looking for schools these will fill the entire beam, just as the bottom normally does. A lot of gain is then not necessary. The **20 log R** setting will provide an acceptable echo strength.

Related topics

- *Echogram* on page 144

Output power

The echo sounder's transducer converts the electrical input power to a sound transmitted into the water. In the majority of the transducers manufactured by Simrad the transducer's *power efficiency* is between 50 and 75%. This means that between 50 and 75% of the input power is transmitted as sound. Transducers from other manufacturers may have as low as 5% power efficiency. Naturally, it is very important that you check this parameter when you wish to purchase a transducer.

The echo sounder's *output power* is a measurement on how much electrical energy the amplifier can send down to the transducer. The maximum power is limited by the transducer you have, and how much power it may accept from the transmitter for conversion to acoustic energy. If you send too much power into the transducer, you may inflict permanent and unrepairable damage.

The *source level (SL)* is a measurement on how much acoustic energy that is in fact sent out by the transducer, how high "volume" it will emit. The source level is measured as "sound pressure" one meter below the transducer face, and it is given i *dB re. 1µPa at 1 m*.

In order to know how much power you can use you must know what kind of transducer you are using. Provided that the echo sounder has been installed with a Simrad transducer, and you know what type it is, this is no problem. All necessary parameters about the transducer are then known by the echo sounder, and the software in the sounder will ensure that you do not output too much power. If you use a third party transducer you must manually check that the output power from the Simrad EK60 does not exceed the power rating.

Note

If you send too much power into the transducer it will – just like a loudspeaker – be damaged beyond repair.

If the transducer receives too much power from the echo sounder, it will also *cavitate*. This is a physical phenomenon causing the appearance of gas bubbles immediately below the transducer face. When this happens hardly any energy is sent into the water, and the transducer face is subject to physical damage. The cavitation depends on the power applied, the physical size of the transducer face, how deep the transducer is mounted, and the amount of contamination (air and particles) under the transducer face. Transducers with a large face can accept more power.

> Near sea level, minute bubbles of micron or submicron size are always present in the ocean. When the rarefaction tension phase of an acoustic wave is great enough, the medium ruptures or "cavitates". For sound sources near the sea surface, the ever-present cavitation nuclei permit rupture to occur at pressure swings of the order of 1 atm (0.1 MPa), depending on the frequency, duration, and repetition rate of the sound pulse. Cavitation bubbles may also be produced by Bernoulli pressure drops associated with the tips of high-speed underwater propellers. Natural cavitation is created by photosynthesis.
>
> Several extraordinary physical phenomena are associated with acoustic cavitation. Chemical reactions can be initiated or increased in activity; living cells and macromolecules can be ruptured; violently oscillating bubbles close to a solid surface can erode the toughest of metals or plastics; light may be produced by cavitation (sonoluminescence). The high pressures and high temperatures (calculated to be 30,000° Kelvin) at the inteior during the collapsing phase of cavitating single bubbles can cause emission of a reproducible pulse of light of duration less than 50 picoseconds.
>
> Of direct importance to the use of sound sources at sea is the fact that, as the sound pressure amplitude increases, ambient bubbles begin to oscillate nonlinearly, and harmonics are generated. At sea level, the amplitude of the second harmonic is less than 1 percent of the fundamental as long as the pressure amplitude of the fundamental of a CW wave is less than about 0.01 atm rms (l kPa) (Rusby 1970). This increases to about 5 percent harmonic distortion when the signal is about 10 kPa.

> When the peak pressure amplitude is somewhat greater than 1 atm, the absolute pressure for a sound source at sea level will be less than zero during the rarefaction part of the cycle. In using CW below 10 kHz, this negative pressure, or tension, is the trigger for a sharply increased level of harmonic distortion and the issuance of broadband noise. Any attempt to increase the sound pressure amplitude appreciably beyond the ambient pressure will cause not only total distortion but also the generation of a large cloud of bubbles which will actually decrease the far-field acoustic pressure.
>
> The detailed bubble activities during cavitation have been studied in several laboratories. Acousticians have identified gaseous cavitation resulting in streamers of hissing bubbles that jet away from regions of high acoustic pressure swings, and vaporous cavitation, which radiates shock waves of broadband noise.
>
> — *Herman Medwin & Clarence S.Clay (1998)*[2]

Related topics

- *Changing the pulse duration to enhance the vertical resolution* on page 22
- *Normal Operation* on page 76

Pulse duration

The echo sounder's *pulse duration* is a measurement for how long the acoustic pulse lasts.

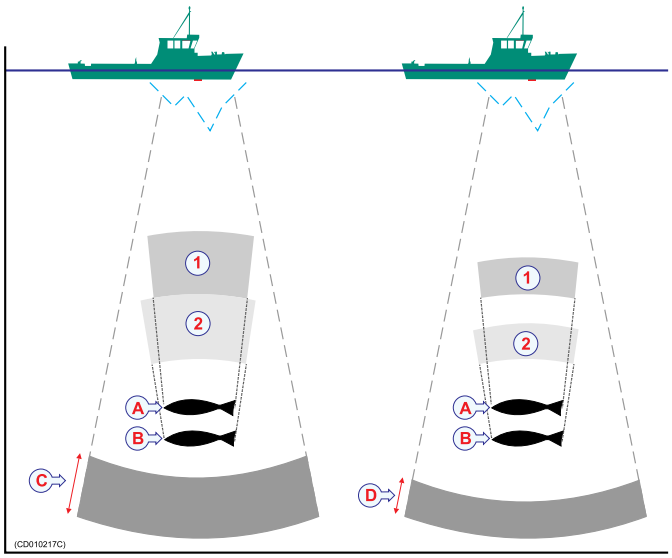
The pulse duration can be adjusted according to the current depth and what kind of fish you are looking for. The deeper you wish to see, the longer pulse duration should be used. Remember that in EK60 echo sounder, the pulse duration and the bandwidth is mutually dependant.

- Long pulse - lots of acoustic energy - narrow bandwidth - less sensitive for noise from own vessel and environment
- Short pulse - less acoustic energy - wide bandwidth - more sensitive for noise from own vessel and environment

2. from “Fundamentals of Acoustical Oceanography”, Academic Press, San Diego, 1998

A pulse duration of 1 mS covers 1,5 meters in the water, and this corresponds to a target separation of approximately 75 cm. This is a typical pulse duration that you may well use down to 250 to 300 meters. If you work in deeper waters use a longer pulse duration, if it is shallower, use a shorter pulse duration. Try out different values, and seek out the pulse duration that provides you with the clearest echo presentation with minimum noise, but with maximum fish detection and separation.

Figure 17 Pulse duration principles



The left vessel uses a long pulse duration (C). As you can see, this causes the echoes from the two fishes (A) and (B) to merge. The right vessel uses a shorter pulse duration, and the two fishes will then appear as two separate echoes on the echogram. Thus, short pulses will provide the best resolution and separation of individual fishes, but the echo sounder is more sensitive to noise.

The speed of sound in water is approximately 1500 m/s. The length of a 1 mS sound pulse will thus be approximately 1,5 meter. With the echo sounder you can then adjust the sound pulse from 7,5 cm (0.05 mS) to 24 m (16 mS) depending of the operational frequency. This is an important factor for the appearance of single fishes.

- When the vertical distance between to fishes, or the distance between a fish and the bottom, is more than the distance covered by a half pulse duration, the echoes will be presented as two separate echoes. The fish above the bottom will be identified.
- If the distance between two individual fishes, or the distance between a single fish and the bottom, is less than the distance covered by a half pulse duration, the echo will be presented as one echo. The echo from the fish close to the bottom will be merged with the bottom echo.

All operational frequencies have different pulse durations. The difference between for example a 50 kHz and a 38 kHz transducer is however not large:

- A 50 kHz transducer can be used with pulse durations from 0,12 mS to 2 mS
- A 38 kHz transducer can be used with pulse durations from 0,26 mS to 4 mS.

Basically, both these frequencies will provide you with the same detection ability. A 50 kHz transducer may provide better resolution in shallow waters, while the 38 kHz transducer may provide longer range on deeper waters. On the 38 kHz transducer the shortest pulse duration is 0,26 mS. This results in a 40 cm sound pulse and a 20 cm fish separation.

Related topics

- *Changing the pulse duration to enhance the vertical resolution* on page 22
- *Normal Operation* on page 76

Range selection

For every frequency (channel) you wish to present on the echo sounder display, you are provided with two echograms.

A *Upper echogram (surface related)*

B *Lower echogram (bottom related)*

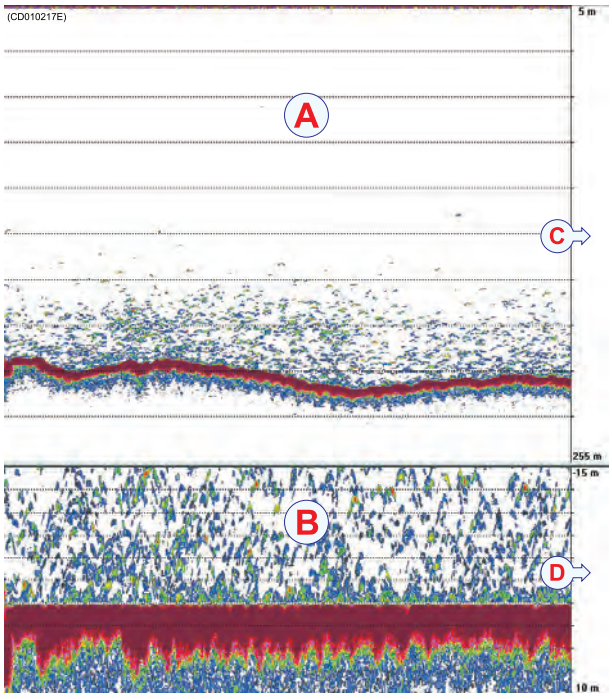
C *Scope view for upper echogram*

D *Scope view for lower echogram*

Normally, the upper echogram will show the entire depth range from the sea surface and down to the bottom, while the lower echogram shows a bottom expansion. This is a magnification of the area just above and below the bottom.

You are free to choose any presentation you want in any of the echograms.

Figure 18 Range selection



Upper echogram: Pelagic or surface related

Example 7 Start Range in a surface related echogram

In a surface echogram, set the **Start Range** value to 0 meters. This will make the echogram start from the sea surface (provided that the transducer offset has been defined). Set **Range** to the current depth plus 20 meters. The echogram will now show the area from the sea surface and down to 20 meters "below" the bottom. The bottom contour is easily detected when the depth changes.

Example 8 Start Range in a surface echogram

In a surface echogram, set the **Start Range** value to 10 meters. This will make the echogram start from 10 meters below the sea surface (provided that the transducer offset has been defined). Set **Range** to the current depth plus 20 meters. The echogram will now show the area from 10 meters below the sea surface, and down to 10 meters "below" the bottom. The bottom contour is easily detected when the depth changes.

Example 9 Start Range in a pelagic echogram

In a pelagic echogram, set the **Start Range** value to 20 meters. This will make the echogram start from 20 meters below the sea surface (provided that the transducer offset has been defined). Set **Range** to 40 meters. The echogram will now show the area from 20 meters below the sea surface, and down to 60 meters below the transducer. Provided that the depth is larger than 60 meters, the bottom contour is not shown.

Lower echogram: Bottom related (bottom expansion)

The majority of our users prefer to use the lower echogram (B) for bottom expansion. To set this up, right-click in the lower echogram to open the **Echogram** dialog, and click **Bottom**.

When you use bottom expansion, the water surface is not the reference any longer, but the bottom is. It is always 0. That means that the bottom in this echogram will always be flat, even though it may vary in the upper echogram. To choose a vertical depth range for the bottom expansion, right-click in the range scale (C) on the right hand side of the echogram to open the **Bottom Range** dialog. Then, select **Range** and **Stop Relative Bottom**. The choice for **Range** defines an area from a few meters below the bottom to a few meters above it. The **Stop Relative Bottom** setting decides how far below or above the area shall stop. In this echogram positive depth is below the bottom, while negative depth is above. These two settings are mutually dependant, so you may turn the bottom line “up side down”. Typical settings may be:

- Range = 15 m
- Stop Relative Bottom = 5 m

This provides you with a 15 meters high phased area starting at 5 meters below the bottom to 10 meters above it.

To change the depth range, use the wheel on the mouse.

Example 10 Start Range and Range in bottom related echogram

In a bottom echogram, set the **Start Range** value to –5 meters. This will make the echogram start from 5 meters above the bottom. Set **Range** to the 5 meters plus 10 = 15 meters. The echogram will now show the area from 5 meters above the depth, and down to 10 meters “below” the bottom. The bottom contour will appear as a flat line.