KONGSBERG

Specifications

EK80 Interface Specifications

Wideband Scientific Echo Sounder

kongsberg.com/ek80

463229/D

PUBLIC

KONGSBERG

EK80

Wide band scientific echo sounder

Interface specifications

24.6.x

The purpose of this manual is to provide the descriptions required to communicate effectively with the EK80 system.

- kongsberg.com/ek80

463229/D

Document information

- **Product**: EK80
- **Document**: Interface specifications
- **Document part number**: 463229
- **Document ISBN number**: N/A
- **Revision**: D
- **Date of issue**: 07 December 2024

Copyright

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. You must be familiar with the contents of the appropriate manuals before attempting to operate or work on the equipment.

Kongsberg Discovery disclaims any responsibility for damage or injury caused by improper installation, use or maintenance of the equipment.

Disclaimer

Support information

If you require maintenance or repair, contact your local dealer. You can also contact us using the following address: support.science@kd.kongsberg.com. If you need information about our other products, visit https://www.kongsberg.com. On this website you will also find a list of our dealers and distributors.

Table of contents

File formats

Topics

File formats supported by the EK80 system

Information collected by the EK80 system can be exported on different file formats.

1. **Raw**: The raw data file format is a proprietary file format by Kongsberg Discovery. It comprises datagrams of XML or binary format. It comprises datagrams of XML or binary format.
2. **Index**: A dedicated index file is created to improve the access to the individual ping data in the raw data file.
3. **XYZ**: This is processed and interpolated "xyz" data in ASCII format. Note that a navigation input must be available.
4. **ADCP NetCDF**: The ADCP netCDF file format is a file format designed by Kongsberg Discovery to hold ADCP velocity data. The format is created as an extension to the ICES SONAR-netCDF4 format using many of the same groups.
5. **BOT**: The BOT file format is a proprietary file format designed by Kongsberg Discovery to contain configuration and depth information.

Related topics

Raw data format, page 135

Index file format for RAW data files, page 180

XYZ file format, page 182

The BOT data file format, page 183

NetCDF file format, page 182

The SEGY data file format, page 185

Raw data format

Related topics

The raw data file format, page 135
Raw data format compatibility, page 140
Using XML datagrams, page 143
Numeric type definition, page 143
Data encapsulation of datagrams, page 141
Configuration XML datagram, page 146
The <Header> tag, page 147
The <Transceivers> tag, page 149
The <Transducers> tag, page 155
The <ConfiguredSensors> tag, page 157
Filter binary datagram, page 164
Environment XML datagram, page 159
Parameter XML datagram, page 171
Sample binary datagram, page 175
Calculating power from Power data recorded with GPT and WBT, page 241
Angle data in Sample datagram, page 245
Special scaling requirements for split beam transducers with three sectors, page 246
NMEA text datagram, page 170
Annotation text (ASCII) datagram, page 145
Motion binary datagram 0, page 168

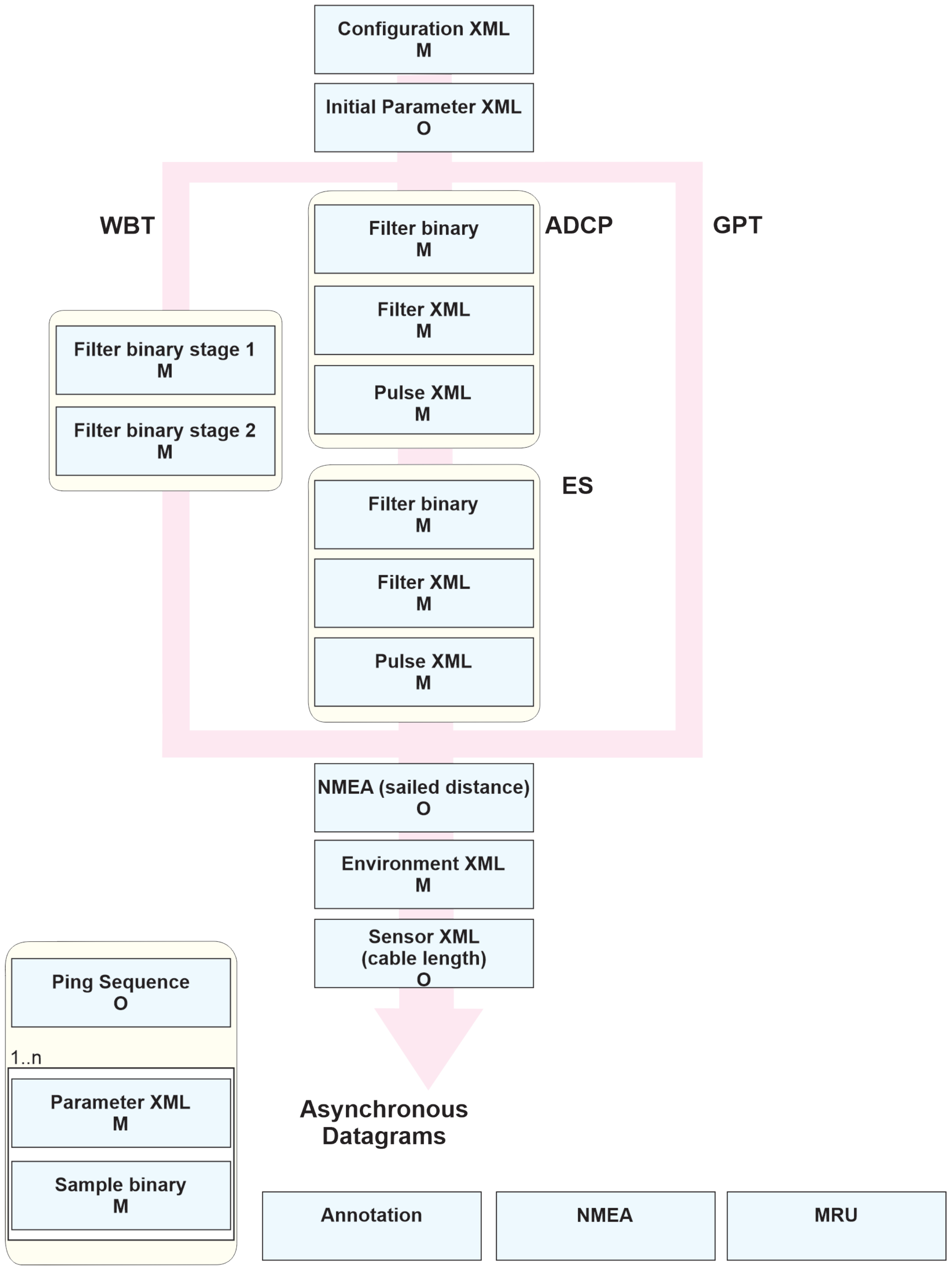
Topics

The raw data file format

Each raw data file contains a set of *datagrams*. The datagrams are in XML, binary or text (ASCII) format. The datagram sequence in the raw data file is not fixed. It depends on the number of installed frequency channels. The files will start with a series of datagrams describing the installed channels. Ahead of the sample datagram sequence there will be a series of NMEA datagram. These are related to the NMEA datagrams received by the EK80 prior to the start of recording. In this context, the term *channel* is used as a common term to identify the combination of transceiver, transducer and operating frequency.

In order to analyse the raw files you will need to keep in mind some general rules of guidance.

- Each raw file starts with a Configuration XML datagram. This is a mandatory datagram. The Configuration XML datagram contains parameters that are not subject to change in the file and describes the system used for this recording.
- Some datagrams are mandatory in a raw file, others are optional. Which datagrams are present in your Simrad system depends on the system configuration.
- The datagram sequence in the raw data file is not fixed. Datagrams describing transceiver configuration are recorded in the installation order of the transceivers. Some datagrams are part of a block, others are received asynchronously.
- Filter and pulse datagrams provide configuration information for a transceiver. Some transceiver types have their own block of datagrams for configuration description. Configuration details of all transceivers installed are recorded.
- Sensor and sample datagrams provide information from each channel and ping. The information in the datagrams are linked using the time stamp and the `Channel ID` information. These datagrams will always be present to provide this information.



The drawing illustrates the datagrams as they may occur in a raw file. The arrows display the sequence of the datagrams. The raw file starts off with a Configuration XML datagram. Which datagrams that follow is dependent on the type of Simrad system, and also the system configuration. The datagrams for each type of transceiver will differ as displayed in *each of the arms in the drawing. The datagrams recorded asynchronous are illustrated at the bottom.*

- O: Optional
- M: Mandatory

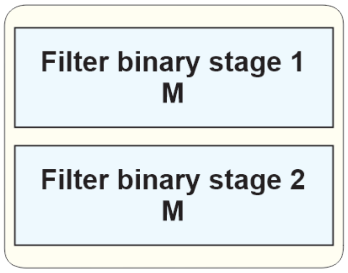
Transceiver configuration details

Some transceiver types have their own block of datagrams for configuration description. Configuration details of all transceivers installed are recorded.

This is a description of the blocks of configuration datagrams related to each transceiver type. Note that the EK80 system may have any number and combinations of transceivers installed.

Wide Band Transceiver (WBT):

EK80 systems with the Wide Band Transceiver (WBT) (and similar) have two Filter datagrams.The first datagram contains the filter parameters from the transceiver, while the second datagram contains the filter parameters from the EK80 program. The two filter datagrams have the same structure. They are referred to as "Stage 1" and "Stage 2".



1. **Filter binary datagram**

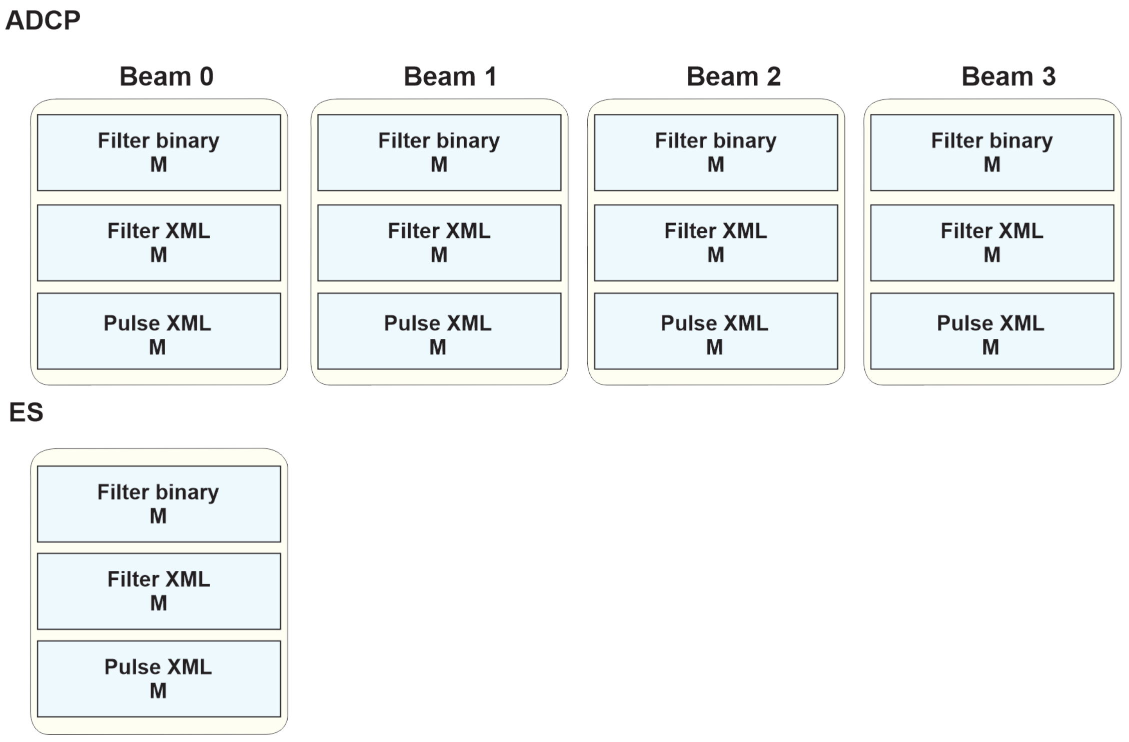
   This is a binary datagram. The type is `FIL1`.

   The Filter binary datagrams contains filter coefficients used for filtering the received signal.

EC150-3C

The EC150–3C configuration details include both current profile (ADCP) mode and echo sounder (ES) mode.

The ADCP configuration consists of the following datagrams. There will be one set of these datagrams for each ADCP beam (four in total).



1. **Filter binary datagram**

   This is a binary datagram. The type is `FIL1`.

   The Filter binary datagrams contains filter coefficients used for filtering the received signal.

2. **Filter XML datagram**

   This is an XML datagram. The type is `XML0`.

   The Filter XML datagram contains parameters for filters.

3. **Pulse XML datagram**

   This is an XML datagram. The type is `XML0`.

   The Pulse XML datagram contains parameters for the transmit pulse.

General Purpose Transceiver (GPT)

Installed General Purpose Transceiver (GPT) do not have Filter datagrams.

The GPT has no specific datagrams transmitted for configuration purposes.

Raw format versions versus the EK80 software versions

Date	Raw format version	EK80 Software version
2014.09.11	1.01	1.8.0 - 1.8.3
2016.02.18	1.10	Not used
2016.06.09	1.20	1.10.0 - 1.11.1
2017.10.02	1.21	1.12.0 - 1.12.1
2018.02.01	1.22	1.12.2 - 1.12.4
2020.07.07	1.23	2.0. - 2.0.1
2022.01.24	1.27	21.15 - 21.15.2
2023.08.01	1.32	23.6
2024.06.10	1.35	24.x

Related topics

Raw data format, page 135
The raw data file format, page 135
Raw data format compatibility, page 140
Using XML datagrams, page 143
Numeric type definition, page 143
Data encapsulation of datagrams, page 141

Raw data format compatibility

The raw data format supported by the Kongsberg EK80 follows the same conventions as the format for the previous Kongsberg EK60.

The format is also common - but not identical - to the raw data formats supported by most other Kongsberg scientific systems. However, in the EK80 format *XML* datagrams are used more frequently, and new datagrams have been introduced.

The EK80 raw data format must be regarded as an *extension* to the EK60 format.

Note:

The EK80 can read the `*.raw` *files generated by the EK60, but the EK60 can not read the EK80 format. Older EK80 software versions may also be unable to read the latest* `*.raw` *files. To ensure compatibility, you must always use the latest EK80 software version.*

The differences between the new and the previous format are all related to the introduction of new parameters and data. In the EK80 format, the XML datagrams contain much of the information found in the EK60 Configuration and Sample datagrams.

Related topics

Raw data format, page 135

The raw data file format, page 135

Raw file datagrams

Related topics

Raw data format, page 135

Data encapsulation of datagrams

A standard encapsulation scheme is used for all datagrams. Each datagram is preceded by a 4-byte length tag stating the datagram length in bytes. An identical length tag is appended to the end of the datagram. The datagram length number is excluding both length tags.

Format

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

Note:

Data encapsulation does not include the NetCDF4 file format.

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 actual content of the datagram may, however, change from one version to the next.

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® operating systems. 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 are specified with this in mind, adding extra unused bytes between fields if necessary.

In case it is necessary, datagrams are padded with extra bytes (from 1 to 3) at the end to ensure that the datagram size is divisible by 4. This is done excluding the length tags that come before and after the datagram itself. This ensures that each datagram and each *length* tag starts at a memory position that is 4-byte aligned. This starting position may be relevant for some processor architectures.

Related topics

Raw data format, page 135

The raw data file format, page 135

Raw file datagrams, page 141

Using XML datagrams

XML datagrams are used to describe parameters and data. These datagrams offer more flexibility than binary datagrams with a fixed structure, and they are not as compressed as binary datagrams are.

XML datagrams are not used for the actual sample data.

The XML datagrams are simply an XML text file following the standard XML convention. Tag attributes are used to contain the information. The length of the XML file can, as with all types of datagrams, be determined from the length of the datagram given in the datagram encapsulation.

The name of the first tag defines the contents of the XML datagram type.

Note:

The Configuration XML datagram replaces the EK60 Configuration datagram (Datagram type CON0).

The Environment XML datagram replaces the environment information part of the EK60 Sample datagram (Datagram type RAW0).

The Parameter XML datagram replaces the parameter information part of the EK60 Sample datagram (Datagram type RAW0).

Related topics

Raw data format, page 135

The raw data file format, page 135

Raw file datagrams, page 141

Numeric type definition

In order to describe the data type formats in the binary datagrams, common "C" structures are used. These represent individual data blocks.

The size of the various "C" types are shown in the table.

"C" type	Content and size
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

Related topics

Raw data format, page 135

The raw data file format, page 135

Raw file datagrams, page 141

Datagram updates

This is a summary of new and updated datagrams for this version of software.

New datagrams

No new datagrams were introduced in the EK80 version 24.6.x.

Updated datagrams

- Configuration XML datagram

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Annotation text (ASCII) datagram

The Annotation text datagram contains comment text that you have typed, for example "Dangerous wreck". It will also contain automatic annotations generated by the EK80. The datagram is inserted whenever a new annotation is generated automatically or manually. The type is `TAG0`.

Format

```
struct TextDatagram
      {
      DatagramHeader DgHeader;
      char Text[];
      };
```

Description

- `DatagramHeader DgHeader`: This is the text (ASCII) datagram in use. The type is `TAG0`.
- `Text[]`: This is the text entered in the annotation.

The text string is zero terminated. The size of the complete datagram depends on the annotation length.

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Configuration XML datagram

The Configuration XML datagram is the first datagram in the raw data file. The Configuration XML datagram contains parameters that are not subject to change in the file and describes the system used for this recording.

Format

The basic XML structure follows. The tags are shown without attributes.

```
<Configuration>
      <Header/>
      <Transceivers>
            <Transceiver>
                  <Channels>
                        <Channel>
                              <Transducer/>
                              <Transducer/>
                        </Channel>
                  </Channels>
            </Transceiver>
      </Transceivers>
      <Transducers>
            <Transducer/>
      </Tranducers>
      <ConfiguredSensors>
            <Sensor/>
      </ConfiguredSensors>
</Configuration>
```

Description

The Configuration XML datagram is identified by the `<Configuration>` tag. The type is `XML0`. Information are contained in the specified attributes. The `<ChannelID>` attribute links data from different datagrams in the raw file to a specific frequency channel.

Related topics

Raw data format, page 135

The <Header> tag, page 147

The <Transceivers> tag, page 149

The <Transducers> tag, page 155

The <ConfiguredSensors> tag, page 157

Raw file datagrams, page 141

Topics

The <Header> tag

The <Header> tag identifies the copyright holder of the raw data format, as well as the application and the relevant versions.

Format

```
<Header
      Copyright="Copyright(c) Kongsberg Maritime AS, Norway"
      ApplicationName="nnn"
      Version="nnn"
      FileFormatVersion="x.x.x"
      TimeBias="yy"
/>
```

Description

The attribute values are only included as examples. The attribute value "nnn" refers to any valid value related to the EK80.

- `ApplicationName`: This attribute identifies the product (EK80).
- `Version`: This attribute displays the EK80 software version.
- `FileFormatVersion`: This attribute identifies the format of the file.

- `TimeBias`: This attribute specifies in which time zone the data was recorded. The time tags in the files are always in Coordinated Universal Time (UTC).

Note:

The version of the raw file format depends on the EK80 software version.

Raw format versions versus the EK80 software versions

Date	Raw format version	EK80 Software version
2014.09.11	1.01	1.8.0 - 1.8.3
2016.02.18	1.10	Not used
2016.06.09	1.20	1.10.0 - 1.11.1
2017.10.02	1.21	1.12.0 - 1.12.1
2018.02.01	1.22	1.12.2 - 1.12.4
2020.07.07	1.23	2.0. - 2.0.1
2022.01.24	1.27	21.15 - 21.15.2
2023.08.01	1.32	23.6
2024.06.10	1.35	24.x

Related topics

Raw data format, page 135

Configuration XML datagram, page 146

Raw file datagrams, page 141

The <ActivePingMode> tag

The `<ActivePingMode>` tag specifies the active operation mode (*Normal* or *Advanced Sequencing*).

Format

These are the attributes in the `<ActivePingMode>` tag. The attribute values are only included as examples.

```
<ActivePingMode Mode="Direct" />
```

Description

The possible values for the Mode attribute are Direct (for *Normal* mode) or Sequence (for *Advanced Sequencing* mode).

Note:

This tag is only used for echo sounders.

Record of changes

1. **1.32**: New

The <Transceivers> tag

The <Transceivers> tag identifies each transceiver that is used on the EK80 system. One or more transceivers can be used, each is specified using a <Transceiver> tag.

Format

The basic XML structure follows. The tags are shown without attributes.

```
<Transceivers>
      <Transceiver>
            <Channels>
                  <Channel>
                        <Transducer/>
                  </Channel>
            </Channels>
      </Transceiver>
</Transceivers>
```

<Transceivers>

The <Transceivers> tag can hold one or more <Transceiver> tags. These are the attributes in the <Transceiver> tag.

The attribute values are only included as examples. The attribute value "nnn" refers to any valid value related to the EK80.

```
<Transceivers
<Transceiver
      TransceiverName="WBT 5197648"
EthernetAddress="009072085343"
IPAddress="172.19.1.114"
Version="[0] Ethernet: 00:90:72:08:53:43
TransceiverSoftwareVersion="1.70"
TransceiverNumber="1"
MarketSegment="nnn"
TransceiverType="nnn"
SerialNumber="nnn"
Impedance="nnn"
Multiplexing="0"
RxSampleFrequency="1500000"
>
<Channels>
</Channels>
</Transceiver>
</Transceivers>
```

<Channels>

Each <Transceiver> tag holds one <Channels> tags. The <Channels> tags holds one or more <Channel> tags. These are the attributes in the <Channels> tag. The attribute values are only included as examples.

```
<Channels>
<Channel
      ChannelID="WBT 545603-15 120-7C_ES"
      LogicalChannelID="WBT 545603-15 120-7C"
      ChannelIdShort="120-7C Drop Keel"
      MaxTxPowerTransceiver="500"
      PulseDuration="6.4E-05;0.000128;0.000256;0.000512;0.001024"
      PulseDurationFM="0.000512;0.001024;0.002048;0.004096;0.008192"
      HWChannelConfiguration="2"
      >
      <Transducer/>
</Channel>
```

```
</Channels>
```

In this context, the term *channel* is used as a common term to identify the combination of transceiver, transducer and operating frequency.

Note:

`PulseDuration` and `PulseDurationFM`may list more or less than 5 values.

<Transducer/>

Each `<Channel>` tag holds one `<Transducer>` tag with attributes. These are the attributes in the `<Transducer/>` tag. The attribute values are only included as examples.

```
<Channel>
<Transducer
      TransducerName="200-7C"
      ArticleNumber="KSV-203003"
      SerialNumber="264"
      Frequency="100000"
      FrequencyMinimum="170000"
      FrequencyMaximum="230000"
      BeamType="0"
      EquivalentBeamAngle="18.5"
      Gain="21.6;22.9;23.1;23.1;23.1
      SaCorrection="0.22;-0.43;-0.66;-0.75;-0.8"
      MaxTxPowerTransducer="500"
      BeamWidthAlongship="9"
      BeamWidthAthwartship="9"
      AngleSensitivityAlongship="0"
      AngleSensitivityAthwartship="0"
      AngleOffsetAlongship="0"
      AngleOffsetAthwartship="0"
      DirectivityDropAt2XBeamWidth="0"
      AdcpBeamAngles="60;180;300"
      AdcpBeamTilt="75"
```

	`AdcpSpeedScaling="0.99"`
	`AdcpCalibratedXOffset="9.0606689453125"`
	`AdcpCalibratedYOffset="5.0606689453125"`
	`AdcpCalibratedZOffset="0.17"`
	`AdcpCalibratedRotationAroundX="1E-05"`
	`AdcpCalibratedRotationAroundY="-0.08095"`
	`AdcpCalibratedRotationAroundZ="8.33225">`
	`<FrequencyPar Frequency="185000" Gain="25.22" Impedance="75" Phase="0" BeamWidthAlongship="7.23" BeamWidthAthwartship="7.47" AngleOffsetAlongship="0" AngleOffsetAthwartship="-0.17">`
	`<FrequencyPar Frequency="185981" Gain="25.21" Impedance="75" Phase="0" BeamWidthAlongship="7.37" BeamWidthAthwartship="7.65" AngleOffsetAlongship="0" AngleOffsetAthwartship="-0.2">`
`</Transducer>`	
`</Channel>`	

Note:

`Gain` and `SaCorrection` may list more or less than 5 values. The number of values listed in the selection is the same as in the `PulseDuration` attribute in the `<Channels>` tag. This means that matching values are created for `PulseDuration`, `Gain` and `SaCorrection` attributes. If you are reading raw files with proprietary software, you must check that the software is capable of handling more than 5 values.

The `BeamType` attribute may have the following values:

- `BeamTypeSingle=0,`

  Standard single beam

- `BeamTypeSplit=1,`

  Standard 4 sector split beam

- `BeamTypeSplit3=17`

  3 sector split beam

- `BeamTypeSplit3C=49`

  3 sectors and a center element split beam

- `BeamTypeSplit3CN=65`

3 sectors and a center element split beam, narrow transmit beam

- `BeamTypeSplit3CW=81`

  3 sectors and a center element split beam, wide transmit beam

- `BeamTypeSplit4B=97`

  4 beam split beam, provided by phased arrays

- `BeamTypeADCPSingle=256`

  Standard 4 beam ADCP

- `BeamTypeADCPSingle3=384`

  3 beam ADCP

`BeamTypeADCPSingle3` attribute is part of the CP200 transducer ADCP.

Attributes beginning with `Adcp` only appear if you are using an ADCP transducer.

- `AdcpBeamAngles` consists of a list of angles in degrees. Each angle is separated by ";".Each angle represents the bearing of the transducer beams referenced to the z-axis.
- `AdcpBeamTilt` is a single angle degrees. The angle represents the beam tilt and the tilt is the same for all beams in the transducer and it is referenced to the XY-plane.
- `AdcpSpeedScaling` represents the scaling factor that needs to be applied to the samples amplitude.
- - `AdcpCalibratedXOffset` represents the X transducer offset taking into account both the installation offset and the calibration performed.
  - `AdcpCalibratedYOffset` represents the Y transducer offset taking into account both the installation offset and the calibration performed.
  - `AdcpCalibratedZOffset` represents the Z transducer offset taking into account both the installation offset and the calibration performed.
- - `AdcpCalibratedRotationAroundX` represents the transducer rotation around X axis taking into account both the installation rotation and the calibration performed.
  - `AdcpCalibratedRotationAroundY` represents the transducer rotation around Y axis taking into account both the installation rotation and the calibration performed.
  - `AdcpCalibratedRotationAroundZ` represents the transducer rotation around Z axis taking into account both the installation rotation and the calibration performed.

The `AngleSensitivity` and `AngleOffset` attributes only appear if you are using a split transducer.

Record of changes

- **1.32**: `<Transducer>` Added:
  - `AdcpCalibratedXOffset`
  - `AdcpCalibratedYOffset`
  - `AdcpCalibratedZOffset`
  - `AdcpCalibratedRotationAroundX`
  - `AdcpCalibratedRotationAroundY`
  - `AdcpCalibratedRotationAroundZ`

  Removed:
  - `AdcpYawCorrection`
  - `AdcpPitchCorrection`
  - `AdcpRollCorrection`
- **1.27**: `<Transducer>` Added:
  - `AdcpBeamAngles`
  - `AdcpBeamTilt`
- **1.23**: `<Transducer>` Added:
  - `AdcpSpeedScaling`
  - `AdcpYawCorrection`
  - `AdcpPitchCorrection`
  - `AdcpRollCorrection`
- **1.22**: `<Transceiver>` Added:
  - `Multiplexing`
  - `RxSampleFrequency`
- **1.20**:
  1. `<Transceiver>` Added:
     - `MarketSegment`
     - `Impedance`
  2. `<Transceiver>` Removed:

- `Parts-list`
- `Product`
- `IP Address`
- `Subnet mask`
- `Default gateway`
- `Embedded software`
- `FPGA TX firmware`
- `FPGA RX firmware`

3. `<Channel>` Added:
    - `PulseDuration`
    - `PulseDurationFM`
    - `BeamTypeSplit3=0x11,`
    - `BeamTypeSplit2=0x21,`
    - `BeamTypeSplit3C=0x31,`
    - `BeamTypeSplit3CN=0x41,`
    - `BeamTypeSplit3CW=0x51`
4. `<Channel>` Removed:
    - `PulseLength`

Related topics

Raw data format, page 135

Configuration XML datagram, page 146

Raw file datagrams, page 141

The <Transducers> tag

The main structure includes a `<Transducers>` tag. The `<Transducers>` tag also includes one or more `<Transducer/>` tags, one for each transducer used on the EK80 system. The

`<Transducer/>` attributes specify the type of transducer, how it is mounted, as well as its physical location in the vessel's coordinate system.

Format

These are the attributes in the `<Transducer/>` tag. The attribute values are only included as examples.

```
<Transducers>
<Transducer
      TransducerName="ES200-7C"
      TransducerMounting="HullMounted"
      TransducerCustomName="ES200-7C Serial No: 1"
      TransducerSerialNumber="1"
      TransducerOrientation="Vertical"
      TransducerOffsetX="0"
      TransducerOffsetY="0"
      TransducerOffsetZ="0"
      TransducerAlphaX="0"
      TransducerAlphaY="0"
      TransducerAlphaZ="0"
      />
</Transducers>
```

Record of changes

- **1.27**: Added: `<TransducerOrientation>`
- **1.20**: Added: `<Transducers>`

Related topics

Raw data format, page 135
Configuration XML datagram, page 146
Raw file datagrams, page 141

The <ConfiguredSensors> tag

The `<ConfiguredSensors>` tag keeps track of the current sensors that are connected to the EK80. The `<ConfiguredSensors>` tag contains one or more `<Sensor>` tags.

Format

The basic XML structure follows. The tags are shown without attributes.

```
<ConfiguredSensors>

      <Sensor>

            <Telegram>

                  |<Value/>

            </Telegram>

      </Sensor>

</ConfiguredSensors>
```

<Sensor>

The `<Sensor>` tag contains attributes with information about the sensor. These are the attributes in the `<Sensor>` tag. The attribute values are only included as examples. The attribute value "nnn" refers to any valid value related to the EK80.

```
<ConfiguredSensor>

<Sensor

      Name="Temperature From LAN Port UDP0"

      Type="Temperature"

      Port="Lan Port 2"

      TalkerID="nnn"

      X="0"

      Y="0"

      Z="0"

      AngleX="0"

      AngleY="0"

      AngleZ="0"

      Unique=”0”

      Timeout="20"

      />

      <Telegram>
```

```
            <Value/>
      </Telegram>
</Sensor>
</ConfiguredSensor>
```

<Telegram>

Each `<Sensor>` tag can contain one or more `<Telegram>` tags. These are the attributes in the `<Telegram>` tag. The attribute values are only included as examples.

```
<Sensor>
<Telegram
      Name="ZDA from GPS From Serial Port 1"
      SensorType="GPS"
      Type="ZDA"
      SubscriptionPath="GPS From Serial Port 1@GPS.TimeInfo"
      Enabled="1"
      >
      <Value/>
</Telegram>
</Sensor>
```

<Value>

Each `<Telegram>` tag can contain one or more `<Value>` tags. These are the attributes in the `<Value>` tag. The attribute values are only included as examples.

```
<Telegram>
<Value
      Name="TimeInfo"
      Priority="1"
      >
</Telegram>
```

Related topics

Raw data format, page 135

Configuration XML datagram, page 146

Raw file datagrams, page 141

Environment XML datagram

The Environment XML datagram contains environment parameters. The absorption coefficient and other related values are calculated using these parameters. This is an XML datagram. The type is `XML0`.

Note:

If the environmental conditions change, the existing raw file will reflect this. The existing raw data file is then automatically closed, and a new file is established with the new environmental data. This ensures that all the data in a single raw data file will always have consistent environmental information.

Format

These are the attributes in the `<Environment>` tag. The attribute values are only included as examples.

```
<Environment
      Depth="100"
      Acidity="8"
      Salinity="35"
      SoundSpeed="1491"
      Temperature="10"
      Latitude="45"
      SoundVelocityProfile="1.000000;1491.000000;1000.000000;14914.000000"
      SoundVelocitySource="Manual"
      TemperatureSource="Manual"
      DropKeelOffset="7.03"
      DropKeelOffsetIsManual="1"
      WaterLevelDraft="23.03"
      WaterLevelDraftIsManual="1"
      TowedBodyDepth="3.01"
      TowedBodyDepthlsManual="1">
      <Transducer
            TransducerName="Unknown"
            SoundSpeed="1500"
            />
```

```
</Environment>
```

Description

The Environment XML datagram is identified by the `<Environment>` tag.

- `Depth`: The value is given in metres.
- `Acidity`: The value is expressed by means of the pH scale.
- `Salinity`: 0 to 40 PSU.
- `SoundSpeed`: The value is given in m/s.
- `Temperature`: This is the water temperature. The value is given in degrees Celsius.
- `Latitude`: The value is given in degrees.
- `SoundVelocityProfile`: The value is given in m/s.
- `SoundVelocitySource`: Calculated / Manual
- `TemperatureSource`:Manual / Probe / Profile / Transceiver
- `TemperatureSourceSelectedTransceiver`: This is the name of the transceiver that has been selected as the temperature source. This attribute is used when "Transceiver" is selected in the `TemperatureSource` attribute.
- `DropKeelOffset`: The value is given in metres.
- `WaterLevelDraft`: The value is given in metres.
- `TowedBodyDepth`: The value is given in metres.
- `TowedBodyDepthlsManual`: The value is given in metres.

The `<Environment>` tag can hold one or more `<Transducer>` elements.The `<Transducer>` tag is provided for future use.

- `TransducerName`: This is normally the actual product name of the transducer.
- `SoundSpeed`: The value is given in m/s.

Record of changes

- **1.35** : `<Environment>` Added:

  `TemperatureSource`, `TemperatureSourceSelectedTransceiver`
- **1.23** : `<Environment>` Added:

  `TowedBodyDepth`, `TowedBodyDepthlsManual`
- **1.20** : `<Environment>` Added:

`Latitude`, `SoundVelocityProfile`, `SoundVelocitySource`, `DropKeelOffset`, `DropKeelOffsetIsManualWaterLevelDraft`, `WaterLevelDraftIsManual`

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Filter XML datagram

The Filter XML datagram contains parameters for filters. This is an XML datagram. The type is `XML0`.

Format

These are the attributes in the `<Filter>` tag. The attribute values are only included as examples.

Example for echo sounder (ES):

```
<Filter ChannelID="EC150 155567-15 EC150-3C - ES_ES"">
      <Stages>
            <Stage1
                  DecimationFactor="24"
                  FilterGeneratorVersion="1.00"
                  CentreFrequency="150000"
                  BandWidth="1944.75"
                  CorrectADSampleTimingInFilter="1"
                  FilterLengthInFilterBanks="480"
                  BitsToTruncateAfterMultiplication="4"
                  FilterGain="1.39139e+06"
            />
            <Stage2
                  DecimationFactor="1"
            />
      </Stages>
</Filter>
```

Description

The Filter XML datagram is identified by the `<Filter>` tag. The `<Filter>` tag holds two `<Stages>` tags.They are referred to as "Stage 1" and "Stage 2".The first datagram contains the filter parameters from the transceiver, while the second datagram contains the filter parameters from the EK80 program. Specific for ADCP there is one Filter XML datagram for each beam, i.e. for four ADCP beams there will be four Filter XML datagrams.

EC150 3C ADCP does not use these additional elements:

- `Notches="0"`
- `TransitionBand="6117.23"`

EC150 3C ADCP does not use these elements:

- `<Stage2>`

Note:

For EC150–3C ADCP there will be one separate datagram for each of the ADCP beams. These are named beam0, beam1, beam2, beam3. Do not interpret these as single beam echo sounder datagrams.

- `Stage`: This is the filter decimation factor.
- `DecimationFactor`: This is the filter decimation factor.
- `CentreFrequency`: The value is given in hertz.
- `BandWidth`: The value is given in hertz.
- `FilterGain`: The value is given in decibels.

Record of changes

1. **1.23** New

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Motion Sensor binary datagram

The Motion Sensor binary datagram includes information regarding heave, roll, pitch and heading.

Format

```
struct StructMotionSensorDatagram
      {
      DatagramHeaderDgHeader;
      float fHeave;
      float fRoll;
      float fPitch;
      float fHeading;
      };
```

Description

- `DatagramHeaderDgHeader`: This is the binary datagram in use. The type is `MRU0`.
- `float fHeave`: This is the vessel movement around the z axis. The value is given in metres.
- `float fRoll`: This is the vessel movement around the x axis. The value is given in degrees.
- `float fPitch`: This is the vessel movement around the y axis. The value is given in degrees.
- `float fHeading`: This is the compass direction of the vessels movement. The value is given in degrees.

Record of changes

1. **1.25**:New

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Filter binary datagram

The Filter binary datagrams contains filter coefficients used for filtering the received signal. The number of Filter datagrams depends on the number of filter stages in the transceiver.

EK80 systems with the Wide Band Transceiver (WBT) (and similar) have two Filter datagrams.The first datagram contains the filter parameters from the transceiver, while the second datagram contains the filter parameters from the EK80 program. The two filter datagrams have the same structure. They are referred to as "Stage 1" and "Stage 2".

Installed General Purpose Transceiver (GPT) do not have Filter datagrams.

Note:

For EC150–3C ADCP there will be one separate datagram for each of the ADCP beams. These are named beam0, beam1, beam2, beam3. Do not interpret these as single beam echo sounder datagrams.

Format

```
struct FilterDatagram
      {
      DatagramHeaderDgHeader;
      short Stage;
      char Spare[2];
      char FilterType;
      char ChannelID[128];
      short NoOfCoefficients;
      short DecimationFactor;
      float Coefficients[];
      };
```

Description

- `DatagramHeaderDgHeader`: This is the binary datagram in use. The type is `FIL1`.
- `Stage`: The is the filter stage number.
- `Spare[2]`: This parameter is provided for future expansions.
- `FilterType`: The is the type of filter.
  - `FilterTypeLowNoise = 0`

    - `FilterTypeLowResolution = 1`
    - `FilterTypeStandardResolution = 2`
    - `FilterTypeHighResolution = 3`
    - `FilterTypeFullResolution = 4`
    - `FilterTypeHydrophone = 5`
    - `FilterTypeHighResolutionReducedAttenuation = 6`
- `ChannelID[128]`: This is the channel identification.
- `NoOfCoefficients`: This is the number of complex filter coefficients.

  The filter coefficients in the Filter datagrams are used in combination with information of the transmitted signal to create the matched filter which can be used to create matched filter or pulse compressed echogram data. The filter coefficients are complex values.

  Thus, the number of values found in `Coefficients[]` are 2 x `NoOfCoefficients` since each complex filter coefficient consist of one real part and one imaginary part. The complex filter coefficients `F(n)` are arranged in `Coefficients[]` as:

  `real(F(1)), imag(F(1)), real(F(2)), imag(F(2)), …. ,`
- `DecimationFactor`: This is the filter decimation factor.
- `Coefficients[]`: These are the filter coefficients.

Record of changes

- **1.21**: Added: `FilterType`

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Initial Parameter XML datagram

The Initial Parameter XML datagram contains information about all available virtual channels that may provide sample data in the file. This is an XML datagram. The type is `XML0`.

Format

These are the attributes in the `<InitialParameter>` tag. The attribute values are only included as examples.

```
<InitialParameter>
      <Channels>
            <Channel PingId="1"
                  ChannelID="WBT 545603-15 ES120-7C_1"
                  ChannelMode="0"
                  PulseForm="1"
                  FrequencyStart="95000"
                  FrequencyEnd="160000"
                  PulseDuration="0.002048"
                  SampleInterval="1.06666666666667E-05"
                  TransmitPower="250"
                  Slope="0.0102796052631579"
                  SoundVelocity="1500"
            />
      </Channels>
</InitialParameter>
```

Description

The Initial Parameter XML datagram is identified by the `<InitialParameter>` tag. The `<InitialParameter>` holds one `<Channels>` tag. The `<Channels>` holds one or more channels, identified by their `ChannelID` tag. The channels have common elements and elements specific to the type of channel or type of pulse form (CW or FM).

A WBT channel contain all of the above elements and no additional elements.

- `ChannelID`: This is the channel identification.
- `ChannelMode`: This is the channel mode. The value 0 indicates active mode, and 1 indicates passive mode.
- `PulseForm`: This indicates the selected pulse form. The value 0 indicates CW (Continuous Wave), and 1 indicates LFM Up (Linear Frequency Modulation).
- `FrequencyStart`: The value is given in hertz.
- `FrequencyEnd`: The value is given in hertz.
- `PulseDuration`: The value is given in seconds.
- `SampleInterval`: The value is given in seconds.

- `TransmitPower`: The value is given in watts.
- `Slope`: The slope value indicates the ratio of the pulse where the transmit power goes from 0 to maximum.
- `SoundVelocity`: The value is given in m/s.

EC150–3C echo sounder channels have these additional elements.

- `Impedance`: The value is given in ohms.
- `Phase`: The value is given in degrees.

EC150–3C ADCP channels have these additional elements.

- `Impedance`: The value is given in ohms.
- `Phase`: The value is given in degrees.

ADCP channels use these additional elements.

- `MaximumVesselSpeed`: 0.0 .. 10.0 m/s
- `MaximumCurrentSpeed`: 0.0 .. 10.0 m/s
- `DepthCellSize: |1|2|4|8| m`

If the `PulseForm` attribute is set to value 0 (zero) for CW, the `FrequencyStart` and `FrequencyEnd` attributes are replaced with a single attribute `Frequency`.

Record of changes

1. **1.23**: New

Related topics

Raw data format, page 135
Raw file datagrams, page 141

Motion binary datagram 0

The Motion binary datagram contains information from the motion reference unit (MRU) or a similar sensor. The datagram is inserted whenever the information from the motion sensor changes. The type is `MRU0`.

Details

From raw file version 1.35: The `MRU0` datagram holds the attitude data transposed to the origo of the vessel array system. This transpose corrects heave for the X-axis and the Y-axis of the attitude sensor and the measured pitch and roll. Pitch and roll values are corrected for any rotation angles of the attitude sensor.

Format

```
struct MRUDatagram
     {
     DatagramHeaderDgHeader;
     float Heave;
     float Roll;
     float Pitch;
     float Heading;
     };
```

Description

- `DatagramHeaderDgHeader`: This is the binary datagram in use. The type is `MRU0`.
- `float Heave`: This is the vessel movement around the z axis. The value is given in metres.
- `float Roll`: This is the vessel movement around the x axis. The value is given in degrees.
- `float Pitch`: This is the vessel movement around the y axis. The value is given in degrees.
- `float Heading`: This is the compass direction of the vessels movement. The value is given in degrees.

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Motion binary datagram 1

The MRU binary datagram contains information from the Motion Reference Unit (MRU) regarding, heave, pitch and roll.This is a binary datagram. The type is `MRU1`.

Format

A standard encapsulation scheme is used for all datagrams.

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

Description

The MRU binary datagram encloses the KM Binary datagram. KM Binary is a proprietary datagram format created by Kongsberg Discovery for general use.

Record of changes

1. **1.23**: New

Related topics

Raw data format, page 135

Raw file datagrams, page 141

NMEA text datagram

The NMEA datagram contains the original NMEA 0183 input message line including carriage return(CR) and line feed (LF). The datagram is inserted whenever the information from one of the relevant sensors changes. The type is `NME0`.

Format

```
struct TextDatagram
      {
      DatagramHeader DgHeader;
      char Text[];
      };
```

Description

- `DatagramHeader DgHeader`: This is the text (ASCII) datagram in use. The type is `NME0`.
- `Text[]`: This is the NMEA message.

The size of the datagram depends on the message length.

Example

NMEA GLL position message:

`$GPGLL,5713.213,N,1041.458,E<cr><lf>`

VTG NMEA VTH speed message:

`$HUVTG,245.0,T,245.0,M,4.0,N,2.2,K<cr><lf>`

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Parameter XML datagram

The Parameter XML datagram contains information about parameters that may change from one transmission ("ping") to the next, for a specific channel. This is an XML datagram. The type is `XML0`.

Format

These are the attributes in the `<Parameter>` tag. The attribute values are only included as examples.

```
<Parameter>

     <Channel

     ChannelID="WBT 545603-15 ES120-7C"

     ChannelMode="0"

     PulseForm="1"

     FrequencyStart="95000"

     FrequencyEnd="160000"

     PulseDuration=”0.002048”

     SampleInterval="1.06666666666667E-05"

     TransmitPower="250"

     Slope="0.0102796052631579"

     SoundVelocity="1492.05"

     />

</Parameter>
```

Description

The Parameter XML datagram is identified by the `<Parameter>` tag. The `<Parameter>` tags can hold one or more `<Channel>` tags.

A WBT channel contain all of the above elements and no additional elements.

- `ChannelID`: This is the channel identification.
- `ChannelMode`: This is the channel mode.
- `PulseForm`: This identifies the selected pulse form out of the different pulse shapes that are available.
- `FrequencyStart`: The value is given in hertz.
- `FrequencyEnd`: The value is given in hertz.

- `PulseDuration`: The value is given in milliseconds.
- `SampleInterval`: The value is given in milliseconds.
- `TransmitPower`: The value is given in watts.
- `Slope`: The **Slope** value identifies how fast the output power in each transmission ("ping") goes from 0 to maximum. The value (in %) indicates the amount of the pulse duration that is spent during this increase.
- `SoundVelocity`: This represents the sound speed at the transducer's face. The value is given in m/s.

When the pulse form is *CW*, `Frequency` is recorded instead of `FrequencyStart` and `FrequencyEnd` attributes.

EC150–3C echo sounder channels have these additional elements.

- `Impedance`: The value is given in ohms.
- `Phase`: The value is given in degrees.
- `EpochTime`: sensor timetag

EC150–3C ADCP channels have these additional elements.

- `Impedance`: The value is given in ohms.
- `Phase`: The value is given in degrees.
- `EpochTime`: sensor timetag

ADCP channels use these additional elements.

- `MaximumVesselSpeed`: 0.0 .. 10.0 m/s
- `MaximumCurrentSpeed`: 0.0 .. 10.0 m/s
- `DepthCellSize: |1|2|4|8| m`

If the `PulseForm` attribute is set to value 0 (zero) for CW, the `FrequencyStart` and `FrequencyEnd` attributes are replaced with a single attribute `Frequency`.

Record of changes

- **1.23** Added:

  `Impedance`, `Phase`, `EpochTime`, `MaximumVesselSpeed`, `MaximumCurrentSpeed`, `DepthCellSize`

- **1.20**

  Added: `PulseDuration`

Removed: `PulseLength`, `TransducerDepth`

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Ping sequence XML datagram

The Ping sequence XML datagram contains information about the virtual channels which will be active in the next transmission ("ping"). This is an XML datagram. The type is `XML0`.

Format

The Ping sequence XML datagram is identified by the `<PingSequence>` tag. The `<PingSequence>` tag can hold one or more `<Ping>` elements containing the `ChannelID` attribute. The attribute values are only included as examples.

```
<PingSequence>

     <Ping ChannelID="EC150 172295-15 EC150-3C - ES_2"/>

     <Ping ChannelID="WBT 518204-15 ES120-7C_3"/>

</PingSequence>
```

Description

Ping sequences are a part of **Mission Planner**. There can be more than one ping sequence comprised in a mission.

- `Ping ChannelID`: This is the channel identification.

Record of changes

1. **1.23**: New

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Pulse XML datagram

The Pulse XML datagram contains parameters for the transmit pulse. This is an XML datagram. The type is `XML0`.

Format

These are the attributes in the `<Pulse>` tag. The attribute values are only included as examples.

Example for echo sounder (ES):

```
<Pulse ChannelID="EC150 155567-15 EC150-3C - ES_ES"
      PulseGeneratorVersion=”1.09”>
      <ESPulses StartDelay=”1.26197e-05”>
            <Pulse1 StartTime="0"
                  Spacing="0"
                  PulseDuration="0.001024"
                  Slope="2.66667e-05"
                  Amplitude="0.6"
                  BandWidth="0"
                  FrequencyStart="150000"/>
                  PSK2="0"
            />
      </ESPulses>
</Pulses>
```

Description

The Pulse XML datagram is identified by the `<Pulse>` tag. The `<Pulse>` tag holds one of two filter type tags, `<ESPulses>` or `<ADCPPulses>`. Specific for ADCP there is one Pulse XML datagram for each beam,i.e. for four ADCP beams there will be four Pulse XML datagrams.

- `Spacing`: This is the minimum acceptable spacing between two targets.
- `PulseDuration`: The value is given in seconds.
- `Slope`: The **Slope** value identifies how fast the output power in each transmission ("ping") goes from 0 to maximum.The value (in %) indicates the amount of the pulse duration that is spent during this increase.
- `Amplitude`: The value is relative (0-1).
- `BandWidth`: The value is given in hertz.
- `FrequencyStart`: The value is given in hertz.

EC150 3C ADCP does not use these additional elements:

- `CorrelationLagInterval="0.0010222`: The value is given in seconds.

Note:

For EC150–3C ADCP there will be one separate datagram for each of the ADCP beams. These are named beam0, beam1, beam2, beam3. Do not interpret these as single beam echo sounder datagrams.

Record of changes

1. **1.23**: New

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Sample binary datagram

The sample datagram contains sample data from each "ping". The datagram may have different size and contain different kinds of data, depending on the DataType parameter. This is a binary datagram. The type is `RAW3`.

Format

```
struct SampleDatagram
      {
      DatagramHeaderDgHeader;
      char ChannelID[128];
      short Datatype;
      char Spare[2];
      long Offset;
      long Count;
      byte Samples[];
      };
```

Description

- `DatagramHeader DgHeader`: This is the binary datagram in use. The type is `RAW3`.
- `ChannelID[128]`: This is the channel identification.
- `Datatype`:
  - Bit 0 = Power

  - Bit 1 = `Angle`
  - Bit 2 = `ComplexFloat16`
  - Bit 3 = `ComplexFloat32`
  - Bit 8 - 10 = `Number of Complex per Samples`
- `Spare[2]`: This parameter is provided for future expansions.
- `Offset`: This is the first sample number.
- `Count`: This is the total number of samples.
- `Samples[]`: These are the received sample values.

  The number of values in `Samples[]` depends on the value of `Count` and the `Datatype`.

  The sample values S(i,n) are arranged as:

  ```
  Real(S(1,1)), Imag(S(1,1)),
  Real(S(2,1)), Imag(S(2,1)),
  Real(S(3,1)), Imag(S(3,1)),
  Real(S(4,1)), Imag(S(4,1)),
  Real(S(1,2)), Imag(S(1,2)), ...
  ```

Example

We have a `DataType` decimal value of 1032.

Decimal number `1032` = Hexadecimal `0408` = Binary `0000 0100 0000 1000`.

- Bit 3 is set to "1", so this is `ComplexFloat32` data.
- Bits 8 to 10 are 100, which means that we have four complex values per sample.

The decimal value of 1032 means that `Samples[]` contains `ComplexFloat32` samples. Each sample consists of four complex numbers (one from each of the four transducer sectors). In this case `Samples[]` consists of 4*2*Count values of 32 bit floats. This is because each sample consists of four complex numbers, and each of these consists of one real part and one imaginary part.

Related topics

Raw data format, page 135

Raw file datagrams, page 141

Transmit pulse sample datagram description

The transmit pulse sample datagram contains sample data from the transmit pulse duration from each "ping".The datagram may have different size and contain different kinds of data, depending on the DataType parameter. This is a binary datagram. The type is `RAW4`. This datagram will always be on complex format. Otherwise refer to RAW3 for Format and Description.

Format

```
struct SampleDatagram
      {
      DatagramHeaderDgHeader;
      char ChannelID[128];
      short Datatype;
      char Spare[2];
      long Offset;
      long Count;
      byte Samples[];
      };
```

Description

- `DatagramHeader DgHeader`: This is the binary datagram in use.
- `ChannelID[128]`: This is the channel identification.
- `Datatype`:
  - Bit 2 = `ComplexFloat16`
  - Bit 3 = `ComplexFloat32`
  - Bit 8 - 10 = `Number of Complex per Samples`
- `Spare[2]`: This parameter is provided for future expansions.
- `Offset`: This is the first sample number.
- `Count`: This is the total number of samples.
- `Samples[]`: These are the received sample values.

  The number of values in `Samples[]` depends on the value of `Count` and the `Datatype`.

  The sample values S(i,n) are arranged as:

```
Real(S(1,1)), Imag(S(1,1)),
Real(S(2,1)), Imag(S(2,1)),
Real(S(3,1)), Imag(S(3,1)),
Real(S(4,1)), Imag(S(4,1)),
Real(S(1,2)), Imag(S(1,2)), ...
```

Example

We have a `DataType` decimal value of 1032.

Decimal number `1032` = Hexadecimal `0408` = Binary `0000 0100 0000 1000`.

- Bit 3 is set to "`1`", so this is `ComplexFloat32` data.
- Bits 8 to 10 are 100, which means that we have four complex values per sample.

The decimal value of 1032 means that `Samples[]` contains `ComplexFloat32` samples. Each sample consists of four complex numbers (one from each of the four transducer sectors). In this case `Samples[]` consists of 4*2*Count values of 32 bit floats. This is because each sample consists of four complex numbers, and each of these consists of one real part and one imaginary part.

When present this datagram always follows a corresponding Parameter XML datagram for a specific channel, in the datagram sequence. All the following conditions must be met before including this parameter in a raw file are.

- The WBT transceiver and the channel must be running in echosounder mode.
- Selected pulse form for the channel is CW, continuous wave.
- The following options are selected in the **Stored sample data for WBTs running CW** in the **File Setup** page in the **Output** dialog box.

  Reduced sampling rate Power/angles samples (Further reduced file size), or **Power/Angle samples (Reduced file size)**

The datagram contains complex samples received corresponding to the transmitted pulse. This ensures more precise information regarding the transmitted pulse when storing power/angle samples instead of complex samples. Storing complex samples calls for larger files and hence the transmit pulse sample datagrams may be a better choice for calculating impendance.

The number of samples contained in the datagram is determined by the pulse duration and sample interval. The relationship between these can be described:

Number of transmit pulse samples = Pulse Duration / Sample interval

Record of changes

1. **1.24**: New

Index file format for RAW data files

A dedicated index file is created to improve the access to the individual ping data in the raw data file. This index file is used when you navigate forward or backwards in the file by means of the Replay bar. The file extension for the index file is `.idx`.

File structure

The index file contains a file header with a Configuration XML datagram, and a number of Index binary datagrams. There is one Index binary datagram for each transmission ("ping").

- The Configuration XML datagram in the header is a copy of the Configuration XML datagram in the corresponding raw data file.
- Each Index binary datagram contains key information about the ping, as well as a pointer (`FileOffset`) to the location of the ping data in the corresponding raw data file.

File structure:

- Configuration XML datagram (The type is `XML0`.)
- *Index binary datagrams with `FileOffset` pointers* (There is one for each transmission ("ping").):
  1. Index binary datagram (The type is IDX0.)
  2. Index binary datagram
  3. Index binary datagram

     ...There is one for each transmission ("ping")....

Format

```
struct IndexFileDatagram
    {
    StructDatagramHeader DgHeader;
    ULONG PingNumber;
    double VesselDistance;
    double Latitude;
    double Longitude;
    DWORD FileOffset;
    };
```

The `FileOffset` pointer

When a raw file is opened, all datagrams up to the `FileOffset` for the first "ping" are read. The `FileOffset` for the first "ping" in the index file points to the first datagram after the file header in the raw data file. This can be any of the following datagrams since they may appear in different order in each "ping".

1. *Sensor data and sample data for each channel and ping:*
    a. Parameter XML datagram
    b. Sample binary datagram
    c. NMEA text datagram (*Asynchronously*)
    d. Annotation text (ASCII) datagram (*Asynchronously*)
    e. Motion binary datagram (*Asynchronously*)

The `FileOffset` pointer for ping "n" will point to the first datagram after the Sample binary datagram for ping "n-1". Consequently, for ping "n", you need the datagrams in the raw file between `fileOffset` in the Index binary datagram number "n" and `fileOffset` in the Index binary datagram number "n + 1". This may be the end of last ping file.

With this change, previous motion reference unit (MRU) and other data will be used for playback of ping "n", instead of the subsequent ones.

This will give the same result if you play back a single ping, or the ping is played back in a sequence of multiple pings. The Parameter XML datagram for ping "n" is always located before the Sample binary datagram, and will therefore always be included in the datagram you read from the raw data file.

The index files will start with XML0 / Version datagram, followed by a list of IDX0 datagrams. With multiple subsequent changes that all indicate that the file should be split, all within a second, you will lose the storage of ping data.

The time format for the files is 1 second resolution. If more files are required within the same second, the file will be overwritten and the last ping data will be stored in the file.

XYZ file format

This is processed and interpolated "xyz" data in ASCII format. The XYZ datagram is a topographical datagram showing the position and depth of a single channel.

Format

```
ll.ddmmhhh,yyyy.ddmmhhh,d.dd,ddMMyyyy,HHmmss.ff,t.ttt,<CR>
```

1. **AA.AAAAAAA**: Latitude in degrees (with its decimals)

   X: North/South

2. **BB.BBBBBBB**: Longitude in degrees (with its decimals)

   Y:

3. **d.dd**: Depth below surface [Metres]
4. **ddMMyyyy**: Date (day, month and year)
5. **HHmmss.ff**: UTC Time
6. **t.ttt**: Transducer offset [Metres]

NetCDF file format

NetCDF (Network Common Data Form) is a data model, API library (application programming interface) and a file format. It is used for storing and managing data. NetCDF is developed and maintained by Unidata. Unidata is part of the US University Corporation for Atmospheric Research (UCAR) Community Programs (UCP). Unidata is funded by the US National Science Foundation.

SONAR-NetCDF4 is a data and metadata convention for storage of data from active sonars in NetCDF4 formatted files, defined by The International Council for the Exploration of the Sea (ICES). Sonar-NetCDF4 consists primarily of a naming convention and a data structure within the NetDCF4 data model. The ADCP-NetCDF4 is based on the SONAR-NetCDF4 version 2.0 and extended with additional data and metadata to store ADCP data.

The echo sounder NetCDF4 information uses SONAR-NetCDF4 version 2.0.

SONAR-NetCDF4 convention version 2.0 has support for storing split beam echosounder data. For the latest update of the convention and the latest approved version please visit:

www.kongsbergdiscovery.net/ek80/index.htm

BOT data format

The BOT data file format

The BOT file format is a proprietary file format designed by Kongsberg Discovery to contain configuration and depth information.

The BOT data file contains a set of *datagrams*. The datagram sequence is not fixed. It depends on the number of installed frequency channels. In this context, the term *channel* is used as a common term to identify the combination of transceiver, transducer and operating frequency.

Depth data for channel and ping

A bot file has the extension .bot. The file contains only one type of datagram, `BOT0`. The BOT0 datagram is encapsulated in the same way as other datagrams used in the raw file format. The first datagram in the file header is followed by depth data in BOT0 datagrams.

The BOT0 datagram provides information from each channel and ping.

Note:

If you have installed six WBTs with split4 transducers, you will have six BOT0 datagram per ping in the file.ent in the bot data files.

Bot data files handling

The characteristics and use of a bot data file is similar to that of the raw data files. The bot data files share these characteristics described for raw data files:

- Data compatibility
- Using XML datagrams
- Numeric type definition

Related topics

BOT data format, page 183
The BOT data file format, page 183
Recording parameters, page 184
Simrad BOT depth datagram format, page 184

Recording parameters

All BOT data recording paramters are specified on the File Setup page. This page is located in the **Output** dialog box. To open, select it on the **Operation** menu.

The **File Setup** settings control how and where the recorded files are saved on the hard disk, or on an external storage device. By adding a prefix to the file names you can identify the files you have recorded during a specific survey.

Tip:

Set up the file and folder parameters before you start the recording. If you wish to save your recorded data on an external hard disk, make sure that it is connected to the computer.

Current Output Folder

Define the output directory for the recorded files. Select **Browse** to choose a different output folder to store the files. This function uses a standard operating system dialog box.

Related topics

BOT data format, page 183

The BOT data file format, page 183

Recording parameters, page 184

Simrad BOT depth datagram format, page 184

Simrad BOT depth datagram format

Simrad BOT is a proprietary datagram format created by Kongsberg. The datagram exports the detected water depth measured from the transducer face to the bottom backscatter value.

Format

```
struct StructDepthDatagram

      {

      StructDatagramHeader DgHeader;

      long lTransducerCount;

      StructChannelDepth Channel[];
      };
```

```
struct StructChannelDepth
      {
      double Depth;
      };
```

Description

- `StructDepthDatagram`
- `DgHeader`
- `lTransducerCount`
- `Channel[]`
- `StructChannelDepth`
- `Depth`

  : Depth [m]

Record of changes

1. **1.23**: New

Related topics

BOT data format, page 183

The BOT data file format, page 183

Recording parameters, page 184

Simrad BOT depth datagram format, page 184

The SEGY data file format

The SEGY (sometimes abbreviated "SEG-Y") file format is one of several standards developed by the *Society of Exploration Geophysicists (SEG)* for storing geophysical data.

It is an open standard, and is controlled by the SEG Technical Standards Committee. For more information, see https://seg.org. Note that a navigation input must be available.

I/O Datagram formats

Related topics

NMEA ZDA datagram format, page 206

Topics

About NMEA and standard datagram formats

Related topics

Topics

About the NMEA datagram formats

The EK80 system can send and receive information to and from several different peripherals. All transmissions take place as *datagrams* with data sentences. Each datagram 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 starts with a “$” delimiter and represents 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 NMEA’s official publications. The *NMEA 1083 - Standard for Interfacing Marine Electronic Devices* document explains the formats in detail. The document can be obtained from NMEA.

Note:

The terms "Datagram" and "telegram" are generally used to describe the basic transfer unit associated with a packet-switched network. The term "sentence" is also used. In this publication, we use the term "datagram".

Related topics

I/O Datagram formats, page 186

About NMEA and standard datagram formats, page 187

NMEA

The National Marine Electronics Association (NMEA) has defined communication standards for maritime electronic equipment. The EK80 system supports these standards for communication with external sensors and peripheral devices.

The most common standard is NMEA 0183. 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 can 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

Related topics

I/O Datagram formats, page 186

About NMEA and standard datagram formats, page 187

NMEA sentence structure

A sentence structure is defined by NMEA to establish the communication between two units. Most other datagram formats are designed using the same, or a similar, structure.

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

```
$aaccc,c-c*hh<CR><LF>
```

$

This character (Hex: 24) is used to identify the start of a sentence.

aaccc

This is the address field. The first two characters (aa) identify 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.

,

The comma (Hex: 2C) is used as a *field delimiter*. 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 that there are no data in the field.

c-c

This is the *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 length, and they are preceded by the field delimiter.

*

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

hh

This is the *checksum*.

<CR><LF>

The carriage return and line feed characters terminate the datagram sentence.

Note:

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

Related topics

I/O Datagram formats, page 186

About NMEA and standard datagram formats, page 187

Standard NMEA 0183 communication parameters

The EK80 system uses both NMEA and proprietary datagram formats to communicate with peripheral systems and sensors. The majority of the datagrams used by the EK80 system are defined by the National Marine Electronics Association (NMEA). NMEA has defined a fixed set of transmission parameters.

The communication parameters defined for NMEA 0183 are:

- **Baud rate**: 4800 bit/s

- **Data bits**: 8
- **Parity**: Even
- **Stop bits**: 1

Some instruments may provide other parameters and/or options. You must always check the relevant technical documentation supplied by the manufacturer.

Related topics

I/O Datagram formats, page 186

About NMEA and standard datagram formats, page 187

NMEA datagram formats

Related topics

NMEA ZDA datagram format, page 206

Topics

NMEA CUR datagram format

The NMEA CUR datagram contains multi-layer water current data. This includes the depth and speed of the current.

Format

```
$--CUR,A,x,d,l.l,m.m,x.x,a,k.k,r.r,h.h,a,a,*hh<CR><LF>
```

Description

This description is not complete. For additional details, refer to the NMEA standard.

1. **$—**: Talker identifier
2. **CUR**: Datagram identifier
3. **A**: Validity
   - **A** = The data are valid.
   - **V** = The data are not valid.
4. **d**: Data set number (1 - 9)
5. **l.l**: Layer number
6. **m.m**: Depth (Metres)
7. **x.x**: Sea current direction in degrees
8. **a**: Direction reference in use:
   - **T** = True
   - **R** = Relative
9. **k.k**: Sea current speed in knots
10. **r.r**: Reference layer depth (Metres)
11. **h.h**: Heading
12. **a**: Heading reference
    - **T** = True
    - **M** = Magnetic
13. **a**: Speed reference
    - **B** = Bottom track
    - **W** = Water track
    - **P** = Positioning system
14. ***hh**: Checksum

Related topics

NMEA datagram formats, page 191

NMEA DBK datagram format

The NMEA DBK datagram contains depth below the keel in feet, meters and fathoms. The datagram is no longer recommended for use in new designs. It is frequently replaced by the NMEA DPT datagram format.

Format

```
$--DBK,x.x,f,y.y,M,z.z,F*hh
```

Description

All depths are measured from below the keel.

1. **$—**: Talker identifier
2. **DBK**: Datagram identifier
3. **x.x,f**: Depth (Feet)
4. **y.y,M**: Depth (Metres)
5. **z.z,F**: Depth (Fathoms)
6. ***hh**: Checksum

Tip:

If you need the depth below the surface, use the NMEA DBS datagram. If you need the depth below the transducer, use the NMEA DBT datagram.

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA DBS datagram format

The NMEA DBS datagram provides the current depth from the surface. The datagram is no longer recommended for use in new designs. It is frequently replaced by the NMEA DPT datagram format.

Format

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

Description

All depths are measured from below the sea surface.

1. **$—**: Talker identifier
2. **DBS**: Datagram identifier
3. **x.x,f**: Depth (Feet)
4. **y.y,M**: Depth (Metres)
5. **z.z,F**: Depth (Fathoms)
6. ***hh**: Checksum

Tip:

If you need the depth below the keel, use the NMEA DBK datagram. If you need the depth below the transducer, use the NMEA DBT datagram.

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA DBT datagram format

The NMEA DBT datagram provides the current depth under the transducer. In new designs, this datagram format is frequently used to replace the DBK and DBS formats.

Format

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

Description

All depths are measured from below the transducer face.

1. **$—**: Talker identifier
2. **DBT**: Datagram identifier
3. **x.x,f**: Depth (Feet)
4. **y.y,M**: Depth (Metres)
5. **z.z,F**: Depth (Fathoms)

6. ***hh**: Checksum

Tip:

If you need the depth below the keel, use the NMEA DBK datagram. If you need the depth below the surface, use the NMEA DBS datagram.

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA DDC datagram format

The NMEA DDC (Display Dimming and Control) datagram format allows you to remotely control the colour palette and brightness of the EK80 display presentations.

Format

```
$--DDC,a,xx,b,c*hh<CR><LF>
```

Description

1. **$—**: Talker identifier
2. **DDC**: Datagram identifier
3. **a**: Display dimming
   - **D** = Daytime setting
   - **K** = Dusk setting
   - **N** = Nighttime setting
   - **O** = The display backlight is turned off.
4. **xx**: Brightness (Percentage)
5. **a**: Colour palette
   - **D** = Daytime setting
   - **K** = Dusk setting
   - **N** = Nighttime setting
   - **O** = The display backlight is turned off.
6. **a**: Status

- **R** = The datagram is provided as a status report.
- **C** = The datagram is provided as a command to change settings.

This datagram description is not complete. For more information, refer to the source specifications issued by National Marine Electronics Association (NMEA).

Related topics

NMEA datagram formats, page 191

NMEA DPT datagram format

The NMEA DPT datagram provides the water depth relative to the transducer, and the offset of the measuring transducer.

Format

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

Description

This description is not complete. For additional details, refer to the NMEA standard.

1. **$—**: Talker identifier
2. **DPT**: Datagram identifier
3. **x.x**: Depth (Metres) Relative to the transducer
4. **y.y**: Offset (Metres) Relative to the 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.
5. **z.z**: This is the maximum range scale in use.
6. ***hh**: Checksum

Tip:

If you need the depth below the keel, use the NMEA DBK datagram. If you need the depth below the surface, use the NMEA DBS datagram. If you need the depth below the transducer, use the NMEA DBT datagram.

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA GGA datagram format

The NMEA GGA datagram transfers 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
```

Description

1. **$—**: Talker identifier
2. **GGA**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
4. **llll.ll,a**: Latitude, North/South (Degrees, minutes and hundredths)
   - **N** = North
   - **S** = South
5. **yyyyy.yy,a**: Longitude, East/West (Degrees, minutes and hundredths)
   - **E** = East
   - **W** = West
6. **x**: Quality indicator for the GPS (Global Positioning System) (Refer to the NMEA standard for further information about the GPS quality indicator.)
7. **zz**: Number of satellites in use: (00 - 12) (The number of satellites may be different from the number in view.)
8. **d.d**: HDOP (Horizontal dilution of precision)
9. **a.a,M**: Altitude related to mean sea level (geoid) (Metres)
10. **g.g,M**: Geoidal separation (Metres)
11. **r.r**: Age of GPS (Global Positioning System) data
12. **cccc**: Identification of differential reference station: (0000 - 1023)
13. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA GGK datagram format

The NMEA GGK datagram is used to decode the PTNL, Time, Position, Type and DOP (Dilution of Precision) string of the NMEA 0183 output.

Format

```
$--
GGK,hhmmss.ss,ddmmyy,nnnnn.nnnnnnnn,a,yyyyy.yyyyyyyy,a,x,zz,w.w,EHTeeeeee,u*hh
<CR><LF>
```

Description

1. **$—**: Talker identifier
2. **GGK**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
4. **ddmmyy**: Day, month and year
5. **nnnnn.nnnnnnnn,a**: Latitude, North/South (Degrees, minutes and hundredths)
   - **N** = North
   - **S** = South
6. **yyyyy.yyyyyyyy,a**: Longitude, East/West (Degrees, minutes and hundredths)
   - **E** = East
   - **W** = West
7. **x**: Quality indicator for the GPS (Global Positioning System) (Refer to the NMEA standard for further information about the GPS quality indicator.)
8. **zz**: Number of satellites in use (00 - 12) (The number of satellites may be different from the number in view.)
9. **w.w**: PDOP (Position dilution of precision)
10. **EHTeeeeee**: Ellipsoidal height of fix
11. **u**: Unit of height measurement
12. ***hh**: Checksum

Related topics

NMEA datagram formats, page 191

NMEA GLL datagram format

The NMEA GLL datagram transfers the latitude and longitude of vessel position, the time of the position fix and the current status from a global positioning system (GPS).

Format

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

Description

1. **$—**: Talker identifier
2. **GLL**: Datagram identifier
3. **llll.ll,a**: Latitude, North/South (Degrees, minutes and hundredths)
   - **N** = North
   - **S** = South
4. **yyyyy.yy,a**: Longitude, East/West (Degrees, minutes and hundredths)
   - **E** = East
   - **W** = West
5. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
6. **A**: Status
   - **A** = The data are valid.
   - **V** = The data are not valid.
7. **a**: Mode indicator
8. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186
NMEA datagram formats, page 191

NMEA HDG datagram format

The NMEA HDG datagram provides heading from a magnetic sensor. If this reading is corrected for deviation, it produces the magnetic heading. If it is offset by variation, it provides the true heading.

Format

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

Description

1. **$—**: Talker identifier
2. **HDG**: Datagram identifier
3. **x.x**: (Degrees (Magnetic))
4. **z.z,a**: Deviation (Degrees (Magnetic)), East/West
   - **W** = West
   - **E** = East
5. **r.r,a** Variation (Degrees (Magnetic)), East/West
   - **W** = West
   - **E** = East
6. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA HDM datagram format

The NMEA HDM datagram provides vessel heading in degrees magnetic. The datagram is no longer recommended for use in new designs. It is often replaced by the NMEA HDG telegram.

Format

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

Description

1. **$—**: Talker identifier

2. **HDM**: Datagram identifier
3. **x.x,M**: (Degrees, Magnetic)
4. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA HDT datagram format

The NMEA HDT datagram provides the true vessel heading. The information is normally provided by a course gyro.

Format

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

Description

1. **$—**: Talker identifier
2. **HDT**: Datagram identifier
3. **x.x,T**: Heading (Degrees, True)
4. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA MTW datagram format

The NMEA MTW datagram provides the current water temperature.

Format

```
$--MTW,x.x,C*hh<CR><LF>
```

Description

1. **$—**: Talker identifier
2. **MTW**: Datagram identifier

3. **x.x,C**: Temperature (Degrees, Celsius)
4. ***hh**: Checksum

Related topics

NMEA datagram formats, page 191

NMEA RMC datagram format

The NMEA RMC datagram transfers the time, date, position, course and speed data from 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
```

Description

1. **$—**: Talker identifier
2. **RMC**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
4. **A**: Status
   - **A** = The data are valid.
   - **V** = The data are not valid.
5. **llll.ll,a**: Latitude, North/South (Degrees, minutes and hundredths)
   - **N** = North
   - **S** = South
6. **yyyyy.yy,a**: Longitude, East/West (degrees, minutes and hundredths)
   - **W** = West
   - **E** = East
7. **x.x**: Speed over ground (knots)
8. **z.z**: Course over ground (degrees (True))
9. **ddmmyy**: Date
10. **r.r,a**: Magnetic variation, East/West (degrees)
    - **E** = East
    - **W** = West

11. **a**: Mode indicator
12. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186
NMEA datagram formats, page 191

NMEA VBW datagram format

The NMEA VBW datagram contains water- and ground-referenced vessel speed data.

Format

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

Description

1. **$—**: Talker identifier
2. **VBW**: Datagram identifier
3. **x.x**: Speed relative to water, Longitudinal (knots)
4. **z.z**: Speed relative to water, Transverse (knots)
5. **A**: Speed relative to water, Status
   - **A** = The data are valid.
   - **V** = The data are not valid.
6. **r.r**: Speed relative to ground, Longitudinal (knots)
7. **q.q**: Speed relative to ground, Transverse (knots)
8. **A**: Speed relative to ground, Status
   - **A** = The data are valid.
   - **V** = The data are not valid.
9. **p.p**: Speed relative to water, Stern, Transverse (knots)
10. **A**: Speed relative to water, Stern, Status
    - **A** = The data are valid.
    - **V** = The data are not valid.
11. **c.c**: Speed relative to ground, Stern, Transverse (knots)

12. **A**: Speed relative to ground, Stern, Status
    - **A** = The data are valid.
    - **V** = The data are not valid.
13. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA VHW datagram format

The NMEA VHW datagram 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>
```

Description

1. **$—**: Talker identifier
2. **VHW**: Datagram identifier
3. **x.x,T**: Heading (Degrees)
4. **x.x,M**: Heading (Degrees (Magnetic))
5. **x.x,N**: Speed relative to water (knots), Resolution = 0.1 knots
6. **x.x,K**: Speed relative to water (km/h), Resolution = 0.1 km/h
7. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA VLW datagram format

The NMEA VLW datagram contains the travelled distance of the vessel. Two values are provided; 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>
```

Description

1. **$—**: Talker identifier
2. **VLW**: Datagram identifier
3. **x.x,N**: Distance relative to water, Total cumulative (nautical miles)
4. **y.y,N**: Distance relative to water, Since reset (nautical miles)
5. **z.z,N**: Distance relative to ground, Total cumulative (nautical miles)
6. **g.g,N**: Distance relative to ground, Since reset (nautical miles)
7. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186
NMEA datagram formats, page 191

NMEA VTG datagram format

The NMEA VTG datagram 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>
```

Description

1. **$—**: Talker identifier
2. **VTG**: Datagram 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)

6. **g.g,K**: Speed over ground (km/h)
7. **a**: Mode indicator
   - A = Autonomous
   - D = Differential
   - N = The data are not valid.
8. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

NMEA ZDA datagram format

The NMEA ZDA datagram contains the universal time code (UTC), day, month, year and local time zone.

Format

```
$--ZDA,hhmmss.ss,xx,yy,zzzz,hh,mm*hh<CR><LF>
```

Description

This description is not complete. For additional details, refer to the NMEA standard.

1. **$—**: Talker identifier
2. **ZDA**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
4. **xx**: Day (01 - 31) (Part of UTC)
5. **yy**: Month (01 - 12) (Part of UTC)
6. **zzzz**: Year (Part of UTC)
7. **hh**: Local time zone, (00 - ±13)
8. **mm**: Local time zone, (00 - 59)
9. ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

NMEA datagram formats, page 191

Proprietary datagram formats

Related topics

Kongsberg CP1 datagram format, page 209

Kongsberg DFT datagram format, page 210

Kongsberg OFS datagram format, page 211

ATS Annotation datagram format, page 212

Simrad EK500 Depth datagram, page 212

Kongsberg EM Attitude 3000 datagram format, page 222

KM Binary datagram format, page 224

Simrad HFB datagram format, page 214

Simrad DBS datagram format, page 215

Simrad TDS datagram format, page 215

Simrad TPR datagram format, page 216

Simrad TPT datagram format, page 217

Simrad PSIMDHB datagram format, page 218

Simrad PSIMP,D datagram format, page 219

Simrad PSIMP,F datagram format, page 221

Topics

Kongsberg CP1 datagram format

The CP1 Current Profile datagram is a proprietary format created by Kongsberg Discovery. The datagram exports the velocity of the water current from the seafloor and from a selection of depth layers in the water column.

Format

```
struct CurrentProfileStruct
{
      public short MessageId;
      public short MessageVersion;
      public ulong PingTime;
      public float StartRange;
      public float SampleInterval;
      public short NumberOfValues;
      public bool IsVesselRelative;
      public bool IsValidBottomData;
      public float BottomVelocityNorth;
      public float BottomVelocityEast;
      public float VesselHeading;
}
```

Start loop (`NumberOfValues`):

```
struct CurrentSampleStruct
{
      public float VelocityNorth;
      public float VelocityEast;
      public bool ValidData;
}
```

End of datagram:

```
      public short Terminator;
```

Description

- **MessageId**: Datagram identifier

- **MessageVersion**: Version of datagram specification (Integer)
- **PingTime**: Time (Time of ping)
- **StartRange**: Start range (Metres)
- **SampleInterval**: Sample interval (Metres)
- **NumberOfValues**: Number of values in datagram
- **IsVesselRelative**: Data relative to vessel (True)
- **IsValidBottomData**: Data from seafloor is valid (True/False)
- **BottomVelocityNorth**: Velocity of the water current at the seafloor (m/s) (Positive value is "North")
- **BottomVelocityEast**: Velocity of the water current at the seafloor (m/s) (Positive value is "East")
- **VesselHeading**: Ship heading (Degrees) (At time of ping)
- **VelocityNorth**: Velocity of the water current at the given depth layer (m/s) (Positive value is "North")
- **VelocityEast**: Velocity of the water current at the given depth layer (m/s) (Positive value is "East")
- **ValidData**: Data from depth layer is valid (True/False)

Related topics

Proprietary datagram formats, page 208

Kongsberg DFT datagram format

The proprietary Kongsberg DFT datagram contains the current water level (draft). The information is required to establish the offset of the transducer face relative to the vessel origin. A custom-built sensor may be required for this measurement.

Many factors can cause the ship's draft to change. The amount of fuel, cargo or ballast may greatly influence the draft. Varying water temperatures and salinity will also have an effect. Draft changes will make any sensor move vertically on the X-axis when referenced to the sea surface. To keep measurements accurate, the location of the water line must therefore be monitored.

Format

```
$KMDFT,xx.xx,*hh
```

Description

The depth is measured relative to the vessel's origin.

- **KM**: Talker identifier
- **DFT**: Datagram identifier
- **xx.xx**: Water level (Metres)
- ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Kongsberg OFS datagram format

The proprietary OFS datagram contains the current length travelled by the drop keel. The information is required to establish the offset of the transducer face relative to the vessel origin. A custom-built sensor may be required for this measurement.

Format

```
$KMOFS,xx.xx, *hh
```

Description

The travelled length is measured as an offset value from the ship origin.

- **KM**: Talker identifier
- **OFS**: Datagram identifier
- **xx.xx**: Depth (Metres)
- ***hh**: Checksum

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

ATS Annotation datagram format

ATS Annotation is a proprietary datagram format created by Kongsberg Discovery. It allows you to import text annotations from external devices.

Format

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

Description

1. **??**: talker identifier
2. **ATS**: datagram identifier
3. **tttt**: free text

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad EK500 Depth datagram

Simrad EK500 Depth is a proprietary datagram format created by Kongsberg Discovery. It was originally defined for the Simrad 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 been designed 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 description

1. **D#**: channel identifier
   - **D1**: Channel 1
   - **D2**: Channel 2
   - **D3**: Channel 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 description

1. **Header#**
   - **D1**: Channel 1
   - **D2**: Channel 2
   - **D3**: Channel 3
2. **Seperator**: ","
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

Related topics

I/O Datagram formats, page 186
Proprietary datagram formats, page 208

Simrad ITI-FS datagram formats

Related topics

Simrad HFB datagram format, page 214
Simrad DBS datagram format, page 215
Simrad TDS datagram format, page 215
Simrad TPR datagram format, page 216
Simrad TPT datagram format, page 217

Simrad HFB datagram format

Simrad HFB is a proprietary datagram format created by Kongsberg Discovery. It provides the vertical distance from the headrope to the footrope (opening), and from the headrope to the bottom (height). The heights are measured by an ITI TrawlEye or a height sensor.

Format

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

Description

1. **II**: Talker identifier
2. **HFB**: Datagram identifier
3. **x.x,M**: This is the distance (height) from the headrope to the footrope. (0 - 100 m)
4. **y.y, M**: This is the distance (height) from headrope to bottom (seabed). (0 - 100 m)

Tip:

If you use two height sensors, the information from the second sensor is provided in the Simrad HB2 datagram.

Related topics

I/O Datagram formats, page 186
Proprietary datagram formats, page 208
Simrad ITI-FS datagram formats, page 213

Simrad DBS datagram format

Simrad DBS is a proprietary datagram format created by Kongsberg Discovery to provide the current depth of the trawl sensor.

Format

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

Description

1. **II**: Talker identifier
2. **DBS**: Datagram identifier
3. **x.x,M**: Depth (Metres) (0 - 2000)

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad ITI-FS datagram formats, page 213

Simrad TDS datagram format

Simrad TDS is a proprietary datagram format created by Kongsberg Discovery to provide the door spread. That is the distance between the two trawl doors.

Format

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

Description

1. **II**: Talker identifier
2. **TDS**: Datagram identifier
3. **x.x,M**: Distance (metres)

Note:

In a dual trawl system, the distance between the second door set is provided in the Simrad TS2 datagram.

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad ITI-FS datagram formats, page 213

Simrad TPR datagram format

Simrad TPR is a proprietary datagram format created by Kongsberg Discovery. It provides the relative bearing and water depth of the trawl sensor, as well as its distance from the vessel. The bearing resolution is 1 degree.

Format

```
@IITPR,x,M,y,P,z.z,M<CR><LF>
```

Description

Note:

All data relate to the trawl sensor (target).

1. **II**: Talker identifier
2. **TPR**: Datagram identifier
3. **x,M**: Horizontal range (Metres) (0 – 4000)
4. **y,P**: Bearing (degrees) (Relative to vessel heading)
5. **z,z,M**: Depth (Metres) (0 - 2000)

Note:

The Simrad ITI measures the depth differently from the range and the bearing. If the ITI only knows the range and the bearing, the depth field is empty.

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad ITI-FS datagram formats, page 213

Simrad TPT datagram format

Simrad TPT is a proprietary datagram format created by Kongsberg Discovery to provide the true bearing and water depth of the trawl sensor, as well as its distance from the vessel. The bearing resolution is 1 degree.

Format

```
@IITPT,x,M,y,P,z.z,M<CR><LF>
```

Description

Note:

All data relate to the trawl sensor (target).

1. **II**: Talker identifier
2. **TPT**: Datagram identifier
3. **x,M**: Horizontal range (Metres) (0 - 4000)
4. **y,P**: Bearing (degrees) (True)
5. **z,z,M**: Depth (Metres) (0 - 2000)

Note:

The Simrad ITI measures the depth differently from the range and the bearing. If the ITI only knows the range and the bearing, the depth field is empty.

Related topics

I/O Datagram formats, page 186
Proprietary datagram formats, page 208
Simrad ITI-FS datagram formats, page 213

Simrad PI50 datagrams

Related topics

Simrad PSIMDHB datagram format, page 218

Simrad PSIMP,D datagram format, page 219

Simrad PSIMP,F datagram format, page 221

Simrad PSIMDHB datagram format

The proprietary Simrad PSIMDHB datagram format is created by Kongsberg Discovery to contain the calculated bottom hardness and biomass information.

Format

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

Description

1. **PS**: Talker identifier
2. **IMDHB**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC)
4. **t**: Transducer number
5. **f,KHZ**: Frequency (kHz)
6. **x.x,M**: Depth (Metres)

   The bottom depth is given as DBS (depth below surface). It is assumed that correct transducer draft has been provided.
7. **y.y,DB**: Bottom hardness (dB)
8. **z.z**: Relative biomass density in m^2/nmi^2 (NASC) (s_A)

   NASC means Nautical Area Scattering Coefficient. This is the format (s_A m^2/nmi^2) in which we provide the biomass data.
9. **,,,**: Spare for future expansions

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad PI50 datagrams, page 217

Simrad PSIMP,D datagram format

Simrad PSIMP ,D is a proprietary datagram format created by Kongsberg Discovery to provide the type and configuration of PS and PI sensors used by a Kongsberg catch monitoring system.

Format

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

Description

Note:

This datagram description is not complete. For more information, contact Kongsberg Discovery.

1. **PS**: Talker identifier
2. **IMP**: Datagram identifier
3. **D**: Sentence specifier
4. **tt**: Time
5. **dd**: Date
6. **M**: Type of measurement
   - **D** = Depth
   - **T** = Temperature
   - **C** = Catch
   - **B** = Bottom
   - **N** = No sensor
   - **M** = Marker
7. **U**: Unit of measurement
   - **M** = Depth measurements (metres)
   - **f** = Depth measurements (feet)
   - **F** = Depth measurements (fathoms)
   - **C** = Temperature measurements (Celsius)
   - **F** = Temperature measurements (Fahrenheit)

8. **S**: Sensor number (1, 2, 3)
9. **C**: Channel (This is the number of the communication channel for the current data source. (1 - 30))
10. **V**: Value (This is the magnitude of the measurement made by the sensor.)
11. **Cr**: Change rate
12. **Q**: Quality
    - **0** = There is no connection between the sensor and the receiver.
    - **1** = One or two telemetry pulses are lost. The measured value is predicted.
    - **2** = The measured data is reliable.
13. **In**: Interference
    - **0** = Interference is not detected.
    - **1** = Interference is detected.
14. **SL**: Signal level (dB // 1 µPa)
15. **NL**: Noise level (dB // 1 µPa)
16. **G**: Gain (0, 20 or 40 dB)
17. **Cb**: Cable quality
    - **0** = The cable is not connected.
    - **1** = The cable is in good working order.
    - **2** = The cable is short-circuited, or the hydrophone current is too large.
18. **error**: Error (**0** means that no errors ares detected. Any other value indicates an error condition.)
19. **chksum**: Checksum (The checksum field consists of a "*" and two hex digits representing the exclusive OR of all characters between, but not including, the "$" and "*" characters.)

Note:

This datagram format is obsolete. It is no longer in use on new designs. It has been replaced by datagram PSIMP ,D1.

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad PI50 datagrams, page 217

Simrad PSIMP,F datagram format

Simrad PSIMP,F is a proprietary datagram format created by Kongsberg Discovery to provide the type and configuration of PS and PI sensors used by a gear monitoring system.

Format

```
$PSIMP,F,S1,S2,S3,T1,T2,T3,F1,F2,F3*chksum<CR><LF>
```

Description

Note:

This datagram description is not complete. For more information, contact Kongsberg Discovery.

1. **PS**:Talker identifier
2. **IMP**: Datagram identifier
3. **F**: Sentence specifier
4. **S1,S2,S3**: Sensor types
   - **D** = Depth (300 metres)
   - **E** = Depth (600 metres)
   - **T** = Temperature
   - **C** = Catch (Slow)
   - **F** = Catch (Fast)
   - **B** = Bottom
   - **N** = No sensor
5. **T1,T2,T3**: Channel
6. **F1,F2,F3**: Offset

7. **chksum**: Checksum (The checksum field consists of a "*" and two hex digits representing the exclusive OR of all characters between, but not including, the "$" and "*" characters.)

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

Simrad PI50 datagrams, page 217

Kongsberg EM Attitude 3000 datagram format

The Kongsberg EM Attitude 3000 is a proprietary datagram format created by Kongsberg Discovery for use with digital motion sensors. It holds roll, pitch, heave and heading information. The datagram contains a 10-byte message.

Format

Data description	Example	Format	Valid range
Sync byte 1 / Sensor status **[1]**	90h to Afh = sensor status	1U	00h, 90h to Afh
Sync byte 2	Always 90h	1U	144
Roll LSB **[2]**		1U	
Roll MSB **[2]**		1U	
Pitch LSB **[2]**		1U	
Pitch MSB **[2]**		1U	
Heave LSB **[2]**		1U	
Heave MSB **[2]**		1U	
Heading LSB **[2]**		1U	
Heading MSB **[2]**		1U	

Description

LSB = least significant byte

MSB = most significant byte.

1. **Sync byte 1 / Sensor status**
   - **00h**: This value is sync byte 1.
   - **90h**: This value indicates valid measurements 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).
- Any value from **A0h** to **AFh** indicates a sensor error status.

2. **All data are in 2's complement binary.**

   Resolution is 0.01 degrees for roll, pitch and heading, and 1 cm for heave.

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

   If a value is outside the valid range, it is assumed to be non-valid, and rejected.

   Note:

   Heave is logged as positive downwards (the sign is changed) including roll and pitch induced lever arm translation to the transmit transducer.

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

Note:

This format was originally designed for use with the early multibeam echo sounders manufactured by Kongsberg Discovery. In the original version of the format (Kongsberg EM Attitude 1000), the first synchronisation byte was always assumed to be zero. The sensor manufacturers were then requested to include sensor status in the format using the first synchronisation byte for this purpose.

Related topics

I/O Datagram formats, page 186

Proprietary datagram formats, page 208

KM Binary datagram format

KM Binary is a proprietary datagram format created by Kongsberg Discovery for general use.

Format

Data description	Unit of measurement	Format	No. of bytes
Start ID	#KMB	char	4U
Datagram length		uint16	2U
Datagram version (=1)		uint16	2U
UTC seconds	s	uint32	4U
UTC nanoseconds	ns	uint32	4U
Status		uint32	4U
Latitude	deg	double	8F
Longitude	deg	double	8F
Ellipsoid height	m	float	4F
Roll	deg	float	4F
Pitch	deg	float	4F
Heading	deg	float	4F
Heave	m	float	4F
Roll rate	deg/s	float	4F
Pitch rate	deg/s	float	4F
Yaw rate	deg/s	float	4F
North velocity	m/s	float	4F
East velocity	m/s	float	4F
Down velocity	m/s	float	4F
Latitude error	m	float	4F
Longitude error	m	float	4F
Height error	m	float	4F

Data description	Unit of measurement	Format	No. of bytes
Roll error	deg	float	4F
Pitch error	deg	float	4F
Heading error	deg	float	4F
Heave error	m	float	4F
North acceleration	m/s^2	float	4F
East acceleration	m/s^2	float	4F
Down acceleration	m/s^2	float	4F
Delayed heave:			
UTC seconds	s	uint32	4U
UTC nanosecond	ns	uint32	4U
Delayed heave	m	float	4F

Description

Data format	Little endian (the least significant byte is transmitted first). Float is according to IEEE - 754.
Datagram length	The total number of bytes in the datagram
Datagram version	The version is incremented if the datagram format is changed.
Timestamp format	Epoch 1970-01-01 UTC time
Position and height	At user-defined sensor reference point. Position in decimal degrees. • Latitude: Negative on Southern hemisphere • Longitude: Negative on Western hemisphere • Height: Positive above ellipsoid
Positive roll	Port side up
Positive pitch	Bow up
Positive heave	Downwards, at user-defined sensor reference point
	True north
Error fields	Sensor data quality: RMS-1= not implemented

Status

One bit per status info, 1= active

Bit	
	Invalid data:
0	Horizontal position and velocity
1	Roll and pitch
2	
3	Heave and vertical velocity
4	Acceleration
5	Delayed heave
Reduced performance:	
16	Horizontal position and velocity
17	Roll and pitch
18	
19	Heave and vertical velocity
20	Acceleration
21	Delayed heave

Related topics

Proprietary datagram formats, page 208

Third-party datagram and file formats

Topics

Atlas Depth datagram format

Atlas Depth is a proprietary datagram format created by Atlas Elektronik (https://www.atlas-elektronik.com) to provide the current depth from two channels.

Format

```
Dyxxxxx.xxm
```

Description

1. **Dy**: Channel number
   - **DA**: Channel number 1
   - **DB**: Channel number 2
2. **xxxxx.xxm**: Depth (Metres)

Related topics

I/O Datagram formats, page 186
Third-party datagram and file formats, page 227

Furuno GPhve datagram format

Furuno GPhve is a proprietary datagram format created by Furuno (http://www.furuno.jp) to contain heave information.

Note:

The datagram format described here remains the property of the organization that defined it. If you need more information, contact the owner.

Format

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

Description

1. **$PFEC**: Talker identifier
2. **GPhve**: Datagram identifier
3. **xx.xxx**: Heave (Metres)
4. **A**: Status
5. ***hh**: Checksum

Description

Positive heave	Down

Related topics

I/O Datagram formats, page 186
Third-party datagram and file formats, page 227

Furuno GPatt datagram format

Furuno GPatt is a proprietary datagram format created by Furuno (http://www.furuno.jp) to contain pitch, roll and yaw information.

Note:

The datagram format described here remains the property of the organization that defined it. If you need more information, contact the owner.

Format

The datagram is available in two versions.

Version 1.5

```
$PFEC,GPatt,xxx.x,yy.y,zz.z<CR><LF>
```

Version 2.0

```
$PFEC,GPatt,xxx.x,yy.y,zz.z*hh<CR><LF>
```

Description

1. **$PFEC**: Talker identifier
2. **GPatt**: Datagram identifier
3. **xxx.x**: Yaw (degrees)
4. **yy.y**: Pitch (degrees)
5. **zz.z**: Roll (degrees)
6. ***hh**: Checksum

Description

Positive roll	Port side up
Positive pitch	Bow up
Positive heading	Clockwise

Related topics

Third-party datagram and file formats, page 227

Hemisphere GNSS GPHEV datagram format

GPHEV is a proprietary datagram format created by Hemisphere GNSS (https://hemispheregnss.com) to contain heave information.

Note:

The datagram format described here remains the property of the organization that defined it. If you need more information, contact the owner.

Format

```
$GPHEV,H,*hh<CR><LF>
```

Description

1. **$**: Talker identifier
2. **GPHEV**: Datagram identifier
3. **H**: Heave (Metres)
4. ***hh**: Checksum

Description

Positive heave	Down

Related topics

I/O Datagram formats, page 186

Third-party datagram and file formats, page 227

Teledyne TSS1 datagram format

Teledyne TSS1 is a proprietary datagram format created by Teledyne TSS Navigation Systems 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.

Format

```
:aabbbb shhhhx srrrr spppp<CR><LF>
```

Description

The format is based on ASCII characters, the datagram has a fixed length, and it is terminated with a carriage return and line feed. Baud rate and output rate may be adjusted to fit your needs. 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:

$$\text{Roll}_{\text{echo sounder}} = \text{arc sin} [\sin(\text{Roll}_{\text{Euler}}) \times \cos(\text{Pitch}_{\text{Euler}})]$$

1. **aa**: Sway acceleration

   This is a dual-character hex number. The value is provided as 0.03835 m/ss units.

2. **bbbb**:Heave acceleration

   This is a four-character hex number. The value is provided as 0.000625 m/ss units.

3. **s**: This is a single character.

   If the value is positive, a "space" character is provided.

   If the value is negative, a "–" character is provided.

4. **hhhh**: Heave position

   This is a four-character decimal number. The value is given in centimetres. Positive value is Up.

5. **x**: Status

   - **U**: Unaided mode/Stable data

     The sensor operates without external input data.

   - **u**: Unaided mode/Unstable data

     The sensor operates without external input data. However, the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on or restarted. The alignment period from a power recycle is normally approximately five minutes.

   - **G**: Speed aided mode/Stable data

     The sensor operates with external input of speed data.

   - **g**: Speed aided mode/Unstable data

     The sensor operates with external input of speed data. However, the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on or restarted. It can also be a failure in the data input.

   - **H**: Heading aided mode/Stable data

     The sensor operates with external input of heading data.

   - **h**: Heading aided mode/Unstable data

     The sensor operates with external input of heading data. However, the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on or restarted. It can also be a failure in the data input.

   - **F**: Full aided mode/Stable data

     The sensor operates with external input of both speed and heading data.

- **f**: Full aided mode/Unstable data

  The sensor operates with external input of both speed and heading data. However, the data from the sensor is unstable. A probable cause for this is the lack of alignment after the sensor has been switched on or restarted. It can also be a failure in the data input.

6. **s**: This is a single character.

   If the value is positive, a "space" character is provided.

   If the value is negative, a "–" character is provided.

7. **rrrr**: Roll angle

   This is a four-character decimal number. The value is given in hundredths of a degree.

8. **s**: This is a single character.

   If the value is positive, a "space" character is provided.

   If the value is negative, a "–" character is provided.

9. **pppp**: Pitch angle

   This is a four-character decimal number. The value is given in hundredths of a degree.

Description

Positive roll	Port side up
Positive pitch	Bow up

Related topics

I/O Datagram formats, page 186

Third-party datagram and file formats, page 227

AML Sound speed datagram format

AML is a third-party proprietary datagram format created by AML Oceanographic (http://www.amloceanographic.com) for use with their sound velocity probes.

Note:

The datagram format described here remains the property of the organization that defined it. If you need more information, contact the owner.

The sound velocity probe output is configurable. The code is searching for a value between 1300 and 1800 and uses it as the sound speed.

The following input data formats are supported by the AML sound velocity probe:

Format

```
xxxx.x<CR><LF>
```

Description

1. **xxxx.x**: Sound speed

Format

```
±xxx.xx yyyy.y<CR><LF>
```

Description

1. **±xxx.xx**: Not used
2. **yyyy.y**: Sound speed

Format

```
dd/mm/yy hh:mm:ss:ms yyy.yy xxxx.xx
```

Description

1. **dd/mm/yy**: Not used
2. **hh:mm:ss:ms**: Not used
3. **yyy.yy**: Not used
4. **xxxx.xx**: Sound speed

Format

```
dd/mm/yy hh:mm:ss:ms xxxx.xx nn.nnn
```

Description

1. **dd/mm/yy**: Not used
2. **hh:mm:ss:ms**: Not used
3. **xxxx.xx**: Sound speed
4. **nn.nnn**: Temperature

Format

```
dd/mm/yy hh:mm:ss:ms yyy.yy xxxx.xx nn.nnn
```

Description

1. **dd/mm/yy**: Not used
2. **hh:mm:ss:ms**: Not used
3. **yyy.yy**: Not used
4. **xxxx.xx**: Sound speed
5. **nn.nnn**: Temperature

Related topics

Third-party datagram and file formats, page 227

Trimble PTNL,GGK datagram format

PTNL ,GGK is a proprietary datagram from Trimble (https://www.trimble.com). The PTNL, GGK datagram is used to decode the time, position, type and dilution of precision of the current position.

Format

```
$PTNL,GGK,hhmmss.ss,ddmmyy,dddmm.mmmmmmmm,a,dddmm.mmmmmmmm,a,x,zz,w.w,E
HTaaa.bbb,M*hh
```

Description

Note:

The datagram format described here remains the property of the organization that defined it. This datagram description is not complete. If you need more information, contact the owner.

1. **$PTNL**: Talker identifier
2. **GGK**: Datagram identifier
3. **hhmmss.ss**: Coordinated Universal Time (UTC) of the current position
4. **ddmmyy**: Day, month and year
5. **dddmm.mmmmmmmm**: Latitude (degrees)

6. **a**: Direction of latitude
   - **N** = North
   - **S** = South
7. **dddmm.mmmmmmmm**: Longitude (degrees)
8. **a**: Direction of longitude
   - **E** = East
   - **W** = West
9. **x**: GPS quality indicator ()
10. **zz**: Number of satellites in use (00 - 12) (The number of satellites may be different from the number in view.)
11. **w.w**: PDOP (Position dilution of precision)
12. **EHTaaa.bbb**: Ellipsoidal height of fix
13. **M**: Ellipsoid height (metres)
14. ***hh**: Checksum

Note:

The PTNL, GGK datagram is longer than the NMEA-0183 standard of 80 characters. The latitude and longitude are in the datum and ellipsoid of the selected reference frame.

GPS quality indicator

Quality indicator 0 (zero) means that fix is not available or invalid. Other values:

1. Autonomous GPS fix
2. Differential, floating carrier phase integer-based solution, RTK (float)
3. Differential, fixed carrier phase integer-based solution, RTK (fixed)
4. Differential, code phase only solution (DGPS)
5. WAAS corrected differential position
6. RTK network position (float solution)
7. RTK network position (fixed solution)

Related topics

Third-party datagram and file formats, page 227

File formats

CTD and sound velocity profile file formats

The EK80 system accepts information from CTD and velocity profilers as files. Several input formats are accepted.

The following formats are supported:

- *.ctd
- *.edf
- *.asvp
- *.vpd
- *.000

Note:

If you observe problems when loading the files, make sure that the file only contains data from surface and down, and not up again. The EK80 system only uses the first part of files containing data from surface and down.

Related topics

File formats, page 133

CTD file format

CTD is a proprietary file format used to store measurement data. The number of columns in the datagram may vary. The first 4 columns are always required: date, time, sound velocity, pressure.

AML CALC file format

AML CALC is a proprietary file format used to contain sound speed profile data. The file format is an ASCII format with a five line header plus a variable number of data lines.

```
CALC, sn, date, depth_increment, depth_display
  AML SOUND VELOCITY PROFILER S/N: xxxxx
```

```
   DATE:xxxxx TIME: xxxxx
   DEPTH OFFSET (M): xxxxx
   DEPTH (M) VELOCITY (M/S) TEMPERATURE (°C)
   0  0  0
```

1. **sn**: Sensor Serial Number

   date: Date

   depth_increment: Logging depth increment

   depth_display: Depth units

2. **S/N**: Sound Velocity Profiler serial number

3. **date**: Julian date

   time: Time when the sensor logging started

4. **depth offset**: Pressure offset at sea level

5. Each line constains depth (m), velocity (m/s) and temperature (°C). Each number is separated by a space character, and the line is terminated with a line feed. All numbers must include a decimal point.

   Example: `xxx.x xxxx.x xx.x <LF>`

6. The last line is used to teminate the information. It must contain three zeros with two space characters between each zero.

   Example: `0 0 0`

Related topics

CTD and sound velocity profile file formats, page 236

Sippican SVP file format

Sippican SVP is a Sippican proprietary format that is used to contain sound speed profile data. The file format is ASCII, and it comprises several lines and comments. This is an example.

```
// This is a MK12 EXPORT DATA FILE (EDF)
//
Date of Launch    : 01/15/2000
Time of Launch    : 17:10:52
Sequence:         : 1
Latitude:         : 43 6.998S
Longitude:        : 148 16.697E
Serial #          : 0
//
```

```
// Here are the contents of the memo fields
//
Mission AUSTREA N/0 L'ATALANTE
//
// Here is some probe information for this drop
//
Probe Type         : T-7
Terminal Depth     : 760 m
Depth Coeff. 1     : 6.691
Depth Coeff. 2     : -0.00225
//
Raw Data Filename : J:\T7_00001.RDF
//
Display Units      : Metrix
//
// This XBT export file has not been noise reduced
// or averaged.
// Sound velocity derived with assumed
// salinity: 30.00 ppt
//
Depth (m) - Temperature (°C) - Sound Velocity (m/s)
0.71    7.32    1508.07
1.3     17.33   1508.12
2.0     17.27   1507.96
2.7     17.28   1508.00
3.3     17.31   1508.11
4.0     17.32   1508.13
4.7     17.32   1508.13
//
// This XCTD export file has not been noise reduced
// or averaged
//
Depth (m) - Temperature (°C) - Salinity (m/s)
0.65    19.31   36.400
1.29    18.83   36.400
1.94    18.60   36.400
2.59    18.51   36.400
```

Related topics

CTD and sound velocity profile file formats, page 236

Valeport file format

Valeport VPD is a proprietary file format used to store measurement data. The file format is an ASCII format with header, columns and data tags with a variable number of data lines.

```
[Application]
Name=Terminal.exe
Title=Valeport Terminal X2
```

```
Version=1.1.0.1315
DateSeparator=.
TimeSeparator=:
ThousandSeparator=
DecimalSeparator=.

[Header]
Count=22
Instrument=MIDAS SVP
Device_Series=400
Device_Type=MIDAS SVP 6000
Serial_Number=62577
Firmware=0400794Z1 08/04/2011 15:00
Latitude=
Longitude=
Site_Info=FACTORY TEST SITE
Sampling_Mode=2 Continuous
Sampling_Rate=4 Hz
Sampling_Period=4 Secs
Average_Type=0 None
Average_Period=4 Secs
File_Number=
Density=1004.700
Gravity=9.807
Battery=11.12 Volts
Pressure_Tare=9.354
Pressure_Tare_Mode=
Pressure_Tare_Time=2021.02.01 15:55:55.000
Time_Stamp=2021.02.04 18:06:23.000
Time_Interval=0.25 Secs

[Calibrations]
Count=3

[Columns]
Count=7
Date/Time=Date/Time;;
Depth=Float;m;Calculated
Sound Velocity=Float;M/SEC;
Pressure=Float;DBAR;
Temperature=Float;C;
Salinity=Float;PSU;Calculated
Density=Float;kg/m³;Calculated

[Data]
Date/Time    Depth    Sound Velocity    Pressure    Temperature
Salinity    Density
    m    M/SEC    DBAR    C    PSU    kg/m³
2021.02.04 18:06:23.000    0.097    0.000    0.097    13.481
```

000 file format

000 is a proprietary file format used to store measurement data. The number of columns in the datagram may vary. The first 4 columns are always required: date, time, sound velocity, pressure.

```
Previous File Location :    0
No of Bytes Stored in Previous File :    0
Model Name :    xxx
File Name :    C:\xxx
Site Information :    xxx
Serial No. :    xxx
No. of Modules Connected :
Fitted Address List :
Parameters for each module :
User Calibrations :
Secondary Cal Used :
Gain :
Offset :
Gain Control Settings :
SD Selected Flag :
Average Mode :    NONE
Moving Average Length:    1
Sample Mode :    TRIP
Sample Interval :    2
Sample Rate :    8
Sample Period :    20
Tare Setting :    xxx
Tare Time Stamp :
Density :
Gravity :
Time Stamp :    dd/mm/yyyy hh:mm:ss
External PSU Voltage :    23.639
Date / Time    SOUND VELOCITY;M/SEC    PRESSURE;DBAR
TEMPERATURE;C    CONDUCTIVITY;MS/CM
23/08/2024 15:46:55    1500.000    1.022    31.154    56.588
23/08/2024 15:46:59    1546.881    2.050    31.091    58.146
23/08/2024 15:47:03    1546.764    3.065    31.121    58.142
23/08/2024 15:47:08    1546.727    4.082    31.152    58.188
23/08/2024 15:47:09    1546.754    5.049    31.155    58.198
23/08/2024 15:47:13    1546.759    6.057    31.159    58.208
23/08/2024 15:47:16    1546.778    7.005    31.159    58.224
23/08/2024 15:47:20    1546.796    8.040    31.161    58.230
23/08/2024 15:47:23    1546.817    9.006    31.163    58.240
23/08/2024 15:47:28    1546.872    10.037    31.177    58.292
```

A How to calculate power from power data

Output power for a transducer/transceiver can be calculated using information from the recorded raw files. The **Sample binary datagram** includes power information either directly or as a complex value.

Power data is included if `Datatype` indicates `Power` or `Complex`.

`Datatype`:

- Bit 0 = `Power`
- Bit 1 = `Angle`
- Bit 2 = `ComplexFloat16`
- Bit 3 = `ComplexFloat32`
- Bit 8 - 10 = `Number of Complex per Samples`

Power data for `Complex` and `Power` are processed differently. make sure you use the right processing for the power data received.

Calculating power from `Power` data recorded with GPT and WBT

The power data contained in the sample datagram is compressed. In order to restore the correct value(s), you must decompress the information.

The following equation is used.

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

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

Related topics

Raw data format, page 135
Sample binary datagram, page 175
Configuration XML datagram, page 146

Calculating power from `Complex` data recorded for WBT

The power data from the transducer is provided in a complex format.

Power output for a sample from a WBT can be calculated using the following equation:

$$\text{Power} = N_{segments} \cdot \left(\frac{|Complex|}{2\sqrt{2}}\right)^2 \cdot \left(\frac{z_{transceiver} + z_{transcducer}}{z_{transceiver}}\right)^2 \cdot \frac{1}{z_{transcducer}}$$

- **Power** = power [Watt]
- **N segments** = number of elements in the split beam
- **|Complex|** = magnitude of the complex sample
- **Z transceiver** = impedance for transceiver [Ω]
- **Z transducer** = impedance for transducer [Ω]

Impedance values are retrieved from the Configuration XML datagram.

Related topics

Sample binary datagram, page 175
Configuration XML datagram, page 146

Calculating power from `Complex` data recorded for EC150–3C

The power data from the transducer is provided in a complex format.

In the new version for raw file format file version 1.25, the power calculations for complex data from EC150-3C has been modified. The complex samples contained in Sample binary datagrams (RAW3 type) generated by EC150-3C transducers are now scaled by a scaling factor of 13.3.

The purpose of introducing this scaling factor is to enable using the same equations for power calculations for both wideband transceivers and EC150-3C. The power calculations are based on voltage sample values as for the wideband transceiver complex data.

These new calculations of power replace completely the EC150-3C calculations used in previous versions of **Power calculations from complex data recorded by EC150-3C**. The calculations are described in the following section:

Calculating power from Complex data recorded for WBT, page 242

Related topics

Sample binary datagram, page 175

Configuration XML datagram, page 146

B How to calculate angle from angle data

Split-beam angles, also named angles, for a transducer can be calculated using the recorded raw files. The **Sample binary datagram** includes angle information either directly or as a complex value depending on the transducer geometry and beam type.

A transducer or transceiver has a number of sectors used for transmitting signals. The number and geometry of the sectors will determine the beam type and how to calculate the angles. **BeamType** is a parameter in EK80 which identifies transducer/transceiver type and geometry.

- **Beam Type**: 1

  Geometry: Four quadrants

  Transducer/transceiver example: ES38B
- **Beam Type**: 17

  Geometry: Three sectors

  Transducer/transceiver example: ES38–10
- **Beam Type**: 49, 65, 81

  Geometry: Three sectors and a centre element

  Transducer/transceiver example: ES38–7
- **Beam Type**: 97

  Geometry: Two pairs of sectors perpendicular to each other

  Transducer/transceiver example: EC150–3C

Angle data is provided in the sample binary datagram. The angle data is included if `Datatype` indicates `Angle` or `Complex`.

`Datatype`:

- Bit 0 = `Power`
- Bit 1 = `Angle`
- Bit 2 = `ComplexFloat16`
- Bit 3 = `ComplexFloat32`
- Bit 8 - 10 = `Number of Complex per Samples`

Angle data for `Complex` and `Angle` are processed differently. Make sure you use the right processing for the power data received.

Angle data in Sample datagram

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. The electrical angle must be multiplied with the angle sensitivity to get the target position relative to beam centre. Positive numbers denotes the fore and starboard directions.

Related topics

Raw data format, page 135

Sample binary datagram, page 175

Calculating angle from `Angle` data

Data from the echo sounders are stored on the EK INT Format. If you would like to reuse the angle data, you will have to convert these to Mechanical Angle.

$$u_x = \arcsin\left(\frac{\frac{180}{128}\cdot\varphi_{xEK\text{int}\_Angle}}{\Lambda}\right)$$
$$u_y = \arcsin\left(\frac{\frac{180}{128}\cdot\varphi_{yEK\text{int}\_Angle}}{\Lambda}\right) \quad (1)$$

- **Ux** is the along the transducer.
- **Uy** is the athwart ships angle.
- **Λ** is the *Angle Sensitivity*. This information is normally given in the data sheet for the product. The angle sensitivity is retrieved from the Configuration XML datagram.

Related topics

Sample binary datagram, page 175

Special scaling requirements for split beam transducers with three sectors

`Power/Angle` data from specific split-beam transducers with three sectors require preprocessing.

Processing `Power/Angle` data obtained from Ethernet data subscription or from reading raw files containing `Power/Angle` data, requires special attention. Make sure that you use the correct conversion formulas, in case the data is sampled by a transducer using geometry of the following types.

When you use the Wide Band Transceiver (WBT) with transducers with `BeamType` = 17, 49, 65 or 81 (decimal values), the angle value from `Power/Angle` files must be scaled. After reading and converting the angle values from the file formats to the formats of your own program, do the following scaling:

- `AlongShip angle` must be multiplied by 2/sqrt(3) before calculating arcsin.

- `AthwartShip angle` must be multiplied by 2.

Use the `AngleSensitivityAlongShip` and `AngleSensitivityAthwartship` values as they are. The angle sensitivity is retrieved from the Configuration XML datagram.

Related topics

Raw data format, page 135

Calculating the angles from `Complex` data for **Beam Type**: 1

The split-beam angles, angles for short, can be calculated based on the output angle information included in the Sample binary datagram. Use the beam type of the transducer and the type of angle information in the datagram (`Complex` or `Angle`) to determine the right processing of the angle data

This section describes the steps involved in calculating the angles for a transducer having four sectors. The angle data in the Sample binary datagram is `Complex`. The parameter **Beam Type** will identify the transducer geometry and type. The **Beam Type** for this transducer is provided in the raw file.

- **Beam Type**: 1

This transducer and transducers having the same geometry and sectors will follow the same steps in calculating the angles.

The relationship between the beams, orientation of the beams and numbering of the complex number in the Sample binary datagram is given by the following:

$$\begin{array}{ll} \text{Starboard Aft:} & \mathrm{QUAD}_1 = x_{StrbAft} + \mathrm{j}\,y_{StrbAft} \\ \text{Port Aft:} & \mathrm{QUAD}_2 = x_{PortAft} + \mathrm{j}\,y_{PortAf} \\ \text{Port Fore:} & \mathrm{QUAD}_3 = x_{PortFore} + \mathrm{j}\,y_{PortFore} \\ \text{Starboard Fore:} & \mathrm{QUAD}_4 = x_{StrbFore} + \mathrm{j}\,y_{StrbFore} \end{array} \tag{1}$$

The angles of the split beam, **Ux** and **Uy**, are calculated using the following equation.

$$u_x = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{\left(x_{StrbAft}+x_{PortAft}\right)\left(y_{PortFore}+y_{StrbFore}\right)-\left(x_{PortFore}+x_{StrbFore}\right)\left(y_{StrbAft}+y_{PortAft}\right)}{\left(x_{PortFore}+x_{StrbFore}\right)\left(x_{StrbAft}+x_{PortAft}\right)+\left(y_{PortFore}+y_{StrbFore}\right)\left(y_{StrbAft}+y_{PortAft}\right)}\right)\right)$$

$$u_y = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{\left(x_{PortAft}+x_{PortFore}\right)\left(y_{StrbAft}+y_{StrbFore}\right)-\left(x_{StrbAft}+x_{StrbFore}\right)\left(y_{PortAft}+y_{PortFore}\right)}{\left(x_{StrbAft}+x_{StrbFore}\right)\left(x_{PortAft}+x_{PortFore}\right)+\left(y_{StrbAft}+y_{StrbFore}\right)\left(y_{PortAft}+y_{PortFore}\right)}\right)\right)$$

- **Ux** is the along ships angle.
- **Uy** is the athwart ships angle.
- **Λ** is the *Angle Sensitivity*. This information is normally given in the data sheet for the product. The angle sensitivity is retrieved from the Configuration XML datagram.

The arctan2 is expanded for giving results in the following range: <-Π, Π].

Note:

*Use the section **Implementation of arctan** for implementation details.*

This traditional four quadrant transducer provides the angles in EK INT Format. You will need to transform these to Mechanical Angles for reuse.

Related topics

Implementation of arctan, page 253
Sample binary datagram, page 175
Configuration XML datagram, page 146
The <Header> tag, page 147
The <Transceivers> tag, page 149
The <Transducers> tag, page 155
The <ConfiguredSensors> tag, page 157

Calculating the angles from `Complex` data for **Beam Type**: 17 49, 65 and 81

The split-beam angles, angles for short, can be calculated based on the output angle information included in the Sample binary datagram. Use the beam type of the transducer and the type of angle information in the datagram (`Complex` or `Angle`) to determine the right processing of the angle data

Transducers having three sectors

This section describes the steps involved in calculating the angles for the WBT having three sectors. This transducer and transducers having the same geometry and sectors will follow the same steps in calculating the angles.The angle data in the Sample binary datagram is `Complex`. The parameter **Beam Type** will identify the transducer geometry and type. The **Beam Type** for this transducer is provided in the raw file.

- **Beam Type**:17

This transducer and transducers having the same geometry and sectors will follow the same steps in calculating the angles.

Start by defining the three complex channels from the transducer. In this case the transducer is a WBT.

The following calculations and equations apply when the **BeamType** is “17”.

Starboard Aft:	$TRX_{strb} = x_{strb} + \mathrm{j}\, y_{strb}$
Port Aft:	$TRX_{port} = x_{port} + \mathrm{j}\, y_{port}$
Forward:	$TRX_{forw} = x_{forw} + \mathrm{j}\, y_{forw}$

The angles of the split beam, **Ux** and **Uy**, are calculated using the following equation.

$$u_x = \arcsin\left[\frac{1}{\sqrt{3}\Lambda}\left(\arctan 2\left(\frac{x_{strb} y_{forw} - x_{forw} y_{strb}}{x_{forw} x_{strb} + y_{forw} y_{strb}}\right) + \arctan 2\left(\frac{x_{port} y_{forw} - x_{forw} y_{port}}{x_{forw} x_{port} + y_{forw} y_{port}}\right)\right)\right]$$

$$u_y = \arcsin\left[\frac{1}{\Lambda}\left(\arctan 2\left(\frac{x_{port} y_{forw} - x_{forw} y_{port}}{x_{forw} x_{port} + y_{forw} y_{port}}\right) - \arctan 2\left(\frac{x_{strb} y_{forw} - x_{forw} y_{strb}}{x_{forw} x_{strb} + y_{forw} y_{strb}}\right)\right)\right]$$

- **Ux** is the along ships angle.
- **Uy** is the athwart ships angle.
- **Λ** is the *Angle Sensitivity*. This information is normally given in the data sheet for the product.

The calculated angles are EK INT format. The angles can be converted from EK INT format to Mechanical Angle Format.

The arctan2 is expanded for giving results in the following range: <-Π, Π].

Note:

*Use the section **Implementation of arctan** for implementation details.*

*Use the section **Specal scaling requirements for split beam transducers** to scale the angles appropriately.*

Transducers having three sectors and a centre element

This section describes the steps involved in calculating the angles for the WBT having three sectors and a centre element. The angle data in the Sample binary datagram is `Complex`. The parameter **Beam Type** will identify the transducer geometry and type. The **Beam Type** for this transducer is provided in the raw file.

- **Beam Type**:49, 65, 81

This transducer and transducers having the same geometry and sectors will follow the same steps in calculating the angles.

The relationship between the beams, orientation of the beams and numbering of the complex number in the Sample binary datagram is given by the following:

Starboard Aft:	$TRX_{strb} = x_{strb} + \mathrm{j}\, y_{strb}$
Port Aft:	$TRX_{port} = x_{port} + \mathrm{j}\, y_{port}$
Forward:	$TRX_{forw} = x_{forw} + \mathrm{j}\, y_{forw}$
Centre:	$TRX_{cntr} = x_{cntr} + \mathrm{j}\, y_{cntr}$

The angles of the split beam, **Ux** and **Uy**, are calculated using the following equation.

$$u_x = \arcsin\left[\frac{1}{\sqrt{3}\Lambda}\left(\begin{array}{l}\arctan 2\left(\dfrac{(x_{strb}+x_{cntr})(y_{forw}+y_{cntr})-(x_{forw}+x_{cntr})(y_{strb}+y_{cntr})}{(x_{forw}+x_{cntr})(x_{strb}+x_{cntr})+(y_{forw}+y_{cntr})(y_{strb}+y_{cntr})}\right)\\ +\arctan 2\left(\dfrac{(x_{port}+x_{cntr})(y_{forw}+y_{cntr})-(x_{forw}+x_{cntr})(y_{port}+y_{cntr})}{(x_{forw}+x_{cntr})(x_{port}+x_{cntr})+(y_{forw}+y_{cntr})(y_{port}+y_{cntr})}\right)\end{array}\right)\right]$$

$$u_y = \arcsin\left[\frac{1}{\Lambda}\left(\begin{array}{l}\arctan 2\left(\dfrac{(x_{port}+x_{cntr})(y_{forw}+y_{cntr})-(x_{forw}+x_{cntr})(y_{port}+y_{cntr})}{(x_{forw}+x_{cntr})(x_{port}+x_{cntr})+(y_{forw}+y_{cntr})(y_{port}+y_{cntr})}\right)\\ -\arctan 2\left(\dfrac{(x_{strb}+x_{cntr})(y_{forw}+y_{cntr})-(x_{forw}+x_{cntr})(y_{strb}+y_{cntr})}{(x_{forw}+x_{cntr})(x_{strb}+x_{cntr})+(y_{forw}+y_{cntr})(y_{strb}+y_{cntr})}\right)\end{array}\right)\right]$$

- **Ux** is the along ships angle.
- **Uy** is the athwart ships angle.
- **Λ** is the *Angle Sensitivity*. This information is normally given in the data sheet for the product. The angle sensitivity is retrieved from the Configuration XML datagram.

The calculated angles are mechanical angles.

The arctan2 is expanded for giving results in the following range: <-Π, Π].

Note:

*Use the section **Implementation of arctan** for implementation details.*

*Use the section **Specal scaling requirements for split beam transducers** to scale the angles appropriately.*

Related topics

Implementation of arctan, page 253

Sample binary datagram, page 175

Calculating angle from Angle data, page 246

Configuration XML datagram, page 146

The <Header> tag, page 147

The <Transceivers> tag, page 149

The <Transducers> tag, page 155

The <ConfiguredSensors> tag, page 157

Calculating the angles from `Complex` data for **Beam Type**: 97

The split-beam angles, angles for short, can be calculated based on the output angle information included in the Sample binary datagram. Use the beam type of the transducer and the type of angle information in the datagram (`Complex` or `Angle`) to determine the right processing of the angle data

This section describes the steps involved in calculating the angles for the EC150–3C transducer. The angle data in the Sample binary datagram is `Complex`. The parameter **Beam Type** will identify the transducer geometry and type. The **Beam Type** for this transducer is provided in the raw file.

- **Beam Type**:97

The beams from the transducer is numbered and named according to the numbering in the user interface.

1. Fore Starboard

2. Aft Port
3. Aft Starboard
4. Fore Port

Complex values are retrieved from the Sample binary datagram. These are the real and imaginary parts of the electrical voltage output for each of the individual receiver channels, also referred to as ADCP beams. SEC1 refers to 1 — Fore Starboard and so on.

$$SEC_1 = x_1 + \mathrm{j}\,y_1$$
$$SEC_2 = x_2 + \mathrm{j}\,y_2$$
$$SEC_3 = x_3 + \mathrm{j}\,y_3$$
$$SEC_4 = x_4 + \mathrm{j}\,y_4$$

Temporary angles are calculated using these x and y values.

$$u_{x'} = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{x_2y_1 - x_1y_2}{x_1x_2 + y_1y_2}\right)\right)$$
$$u_{y'} = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{x_4y_3 - x_3y_4}{x_3x_4 + y_3y_4}\right)\right)$$

The arctan2 is expanded for giving results in the following range: <-Π, Π].

Note:

*Use the section **Implementation of arctan** for implementation details.*

Λ is the *Angle Sensitivity*. This information is normally given in the data sheet for the product. The angle sensitivity is retrieved from the Configuration XML datagram.

Some more mathematics and the split-beam angles are given by the following equation.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \frac{\begin{bmatrix} \tan u_{x'} \\ \tan u_{y'} \\ 1 \end{bmatrix}}{\sqrt{\tan^2 u_{x'} + \tan^2 u_{y'} + 1^2}}$$

Place the result from this equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

- Ψ=45°

The angles (**Ux**) and (**Uy**) is calculated using the result from the previous equation.

$$u_x = \arctan 2 \frac{x}{z}$$

$$u_y = \arctan 2 \frac{y}{z}$$

Related topics

Implementation of arctan, page 253
Sample binary datagram, page 175
Configuration XML datagram, page 146
The <Header> tag, page 147
The <Transceivers> tag, page 149
The <Transducers> tag, page 155
The <ConfiguredSensors> tag, page 157

Implementation of arctan

This is the software implementation of the arctan function.

The code stretches over two pages.

```
phi = arctan2(y/x) // phi =< –π,π].          // Normally arcan gives phi
=< –π/2,π/2]{
      if(y=>0)
      {
            if(x>0)
            {
                  phi=arctan(y/x) //First Quadrant
            {
```

```
        elseif(x<0)
        {
            phi=π + arctan(y/x) //Second Quadrant
        }
        else
        {
            phi = π/2 //x=0, y=1)
        }
    }
    elseif(y<0)
    {
        if(x<0)
        {
            phi = −π + arctan(y/x) //Third Quadrant
        }
        elseif(x>0)
        {
            phi = arctan(y/x) //Fourth Quadrant
        }
        else
        {
            phi = −π/2 //(x=0, y=-1)
        }
    }
    else

    {
        if(x=-1)
        {
                phi = π //(x=-1, y=0)
            }
            else
            {
                phi =0 //(x=1, y=-0)
            }
        }
}
```

Related topics

Sample binary datagram, page 175

C Calculating transducer impedance with WBT

Transducer impedance with WBT using RAW3

Calculating transducer impedance with WBT involves looking at data from the Sample binary datagram.

Calculating transducer impedance with WBT

- The first 16–bits represent current measurements.
- The next 16–bits represent voltage measurements.

By adding zeros to the last 16–bit, mantissa bits, two float values can be obtained. Each complex sample will in this way be split into a complex current value and a complex voltage value, ready to be used for calculation of transducer impedance.

These special format values are outputted as long as one of the WBT channels are transmitting. The number of values to be used for impedance calculation must be based on the information in the Filter XML datagrams and the pulse duration.

Number of samples to remove before starting to calculate the transducer impedance in BITE:

- N1 = *Stage1 filter NoOfCoefficients*
- N2 = *Stage2 filter NoOfCoefficients* in Stage2

- D1 = *Stage1 filter DecimationFactor*
- D2 = *Stage2 filter DecimationFactor*

NoOfCoefficients is an element in the Filter binary datagram.

DecimationFactor is an element in the Filter XML datagram.

Calculating total filter delay use this formula:

```
Total filter delay = (N1/2/D1 + N2/2) / D2
```

The calculation is valid for the start sample and the number of samples within the pulse duration.

- For Frequency Modulated (FM) pulse formats the raw files include a certain number of extra samples. These extra samples must be removed, before calculating the transducer impedance.
- For signals using Continuous Wave (CW) pulse format, the raw file recordings does not include extra samples. These have been removed before data was stored in raw files. You do not have to remove any additional samples.

WBT/EK80 sample data

Sample format from WBT during Tx/Rx

The sample data format for WBT is not the same during transmission and reception. The following to sections describe what the sample data looks like during these two functions.

Each sample data consists of a 32–bit section. This section consists of two 16–bit subsections.

- The first 16–bits represent current measurements.
- The next 16–bits represent voltage measurements.

Transmission

During transmission the sample data contains current and voltage data infomration. The current and voltage sample data are comprised by two parts, the real and imaginary part. The real part of both current and voltage is found in the first 32–bit field. The imaginary part of current and voltage is found in the second 32–bit field.

The first 32–bits:

Current real part																**Voltage real part**															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

The second 32–bits:

Current imaginary part																**Voltage imaginary part**															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

- S: sign
- e: exponent
- m: mantissa

Reception

During reception the sample data contains high and low voltage gain data information. The high and low voltage gain data are comprised by two parts, the real and the imaginary part. The real part of both high and low voltage gain is comprised in the first 32–bit field. The imaginary part of high and low voltage is found in the second 32–bit field.

The first 32–bits:

Voltage High Gain real part																**Voltage Low Gain real part**															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

The second 32–bits:

Voltage High Gain imaginary part																**Voltage Low Gain imaginary part**															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

- S: sign
- e: exponent
- m: mantissa

Sample range for current/voltage samples

The number of current/voltage samples made during transmission is calculated. The base of this calculation is the maximum value of the pulse length, filter length and any pulse delay.

The maximum pulse duration for the different types of WBTs are:

- WBT: 29.1 ms
- WBT HP: 52.4 ms
- WBT Mini: 25.7 ms

Selection of high/low gain samples

The gain selector is a tool used to select sample voltage level selection. Low gain is automatically selected for all values when the low gain voltage exceeds 3 mV (peak value).

Both the selected and the not selected gain are sent/saved. Tools like oscilloscope view and reprocessing the gain selected in replay.

From EK80 v.1.10.1 re-selection of gain during replay is default.

Transmission

The first 32–bits:

Voltage real part																Current real part															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

The second 32–bits:

Voltage imaginary part																Current imaginary part															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

- S: sign
- e: exponent
- m: mantissa

Reception

The first 32–bits:

Selected real part																The other real part															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m

The second 32–bits:

Selected imaginary part																The other imaginary part															
S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	m	S	e	e	e	e	e	e	e	e	m	m	m	m	m	m	X

- S: sign
- e: exponent
- m: mantissa
- X: This last bit in the imaginary value is telling what type of gain has been selected.

1 represents high gain, 0 represents low gain.

Raw data content

The raw data files contain all samples, on this format after selection of gain chain.

From EK80 v.1.10.1 (20161122) gain is reselected by default during replay.

Related topics

Sample binary datagram, page 175

Transducer impedance with WBT using RAW4

Calculating transducer impedance with WBT involves looking at data from the Sample binary datagram. This description is valid for RAW4.

Calculating transducer impedance with WBT

Calculating transducer impedance with WBT (wideband transceiver) involves looking at data from the Sample binary diagram. To do this using RAW4 datagram format, use the description for RAW3 format. Note that the RAW4 datagram format will only contain the current/voltage samples during transmission.

Index

Copyright