KONGSBERG

INTERFACE SPECIFICATIONS

Kongsberg EK80

kongsberg.com/ek80

PUBLIC

KONGSBERG

EK80

Wide band scientific echo sounder

Interface specifications

Release 23.6.0

463229/C

Document information

- **Product**: Kongsberg EK80
- **Document**: Interface specifications
- **Document part number**: 463229
- **Document ISBN number**: N/A
- **Revision**: C
- **Date of issue**: 17 August 2023

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

Disclaimer

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

Support information

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

Kongsberg Maritime AS
www.kongsberg.com

Table of contents

About this manual

This manual is intended for all users of the EK80 systems that need to create remote control and/or data acquisition applications. It is also intended for those using the output files provided by the EK80 system for further processing and analysis. Due to the nature of the descriptions and the level of detail provided by this publication, it is well suited for those who are - or wish to be - expert users.

This manual is a reference book, and as such it is *not* intended for sequential reading. Use the table of content, the index, the search functionality as well as the interactive links to seek out the information you need when you need it.

A good understanding of system functions and controls is essential to fully take advantage of the functionality provided.

Online information

All end-user manuals provided for operation and installation of your EK80 system can be downloaded from our website.

- kongsberg.com/ek80

License information

To obtain a license, contact your local dealer.

Software version

This EK80 Interface specifications complies with software version 23.6.0.

Registered trademarks

Observe the registered trademarks that apply.

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

Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.

We want your feedback

We always want to improve our products. We also want our end-user documentation to be comprehensive and relevant. You can help. Please provide comments, suggestions or constructive criticism to any of our support offices.

Data subscriptions and remote control

Topics

About data subscriptions and remote control

The EK80 Wide band scientific echo sounder allows you subscribe to echo data and to control the operation from your own remote application. In this way you can create your own software for controlling the EK80 operations. Such operations may for example include start/stop pinging, changing ping interval, or start/stop data recording.

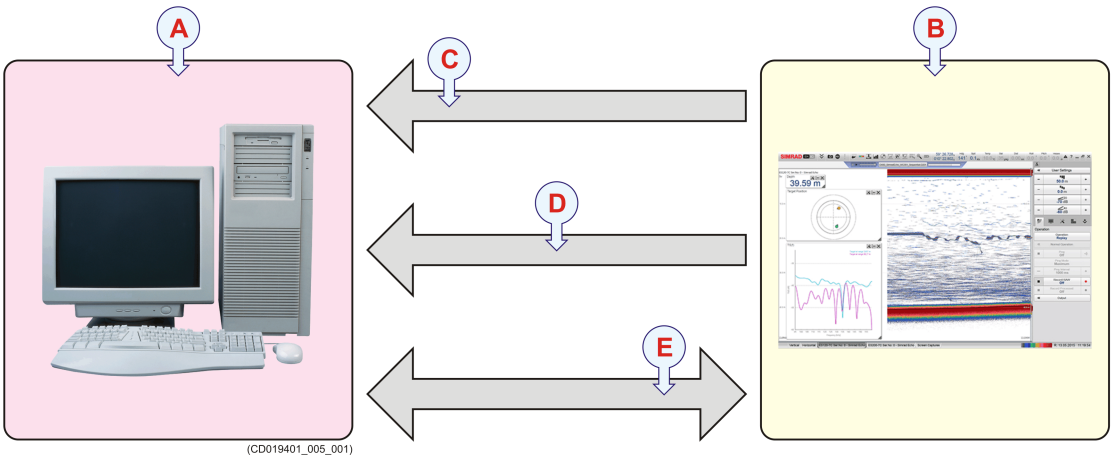
By means of your client application you can subscribe to data from the EK80. This means that you can ask the EK80 to continuously send various data (for example Depth data, Target Strength data or Integration data) to your client application.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

The communication between the EK80 program and your own client application is done by exchanging UDP messages via the local area network (LAN). Command and response messages are all text messages on XML format. Subscribed data updates are binary data structures. These must be decoded using the relevant information about the data structure.

> UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.
>
> With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level.
>
> *https://en.wikipedia.org/wiki/User_Datagram_Protocol, April 2016*



A Local computer (normally connected to the local area network) running your own program ("client application")

B Processor Unit running the EK80 program ("server application")

C Subscriber parameter updates (UDP/Binary)

D Subscribed data updates (UDP/Binary)

E Commands and responses (UDP/XML)

Note

For new data subscription and remote control applications, we recommend using the REST API.

REST API, page 103

Overview of the data subscriptions

A specific process must be implemented to establish data subscription and remote control.

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

1 Request server information

2 Connect to server

3 Keep connection alive

4 Issue commands to the server

Data subscription processes:

a Create data subscription

b Handling data

c Change data subscription

d Destroy data subscription

Parameter management:

a Get parameter

b Set parameter

c Get parameter information (attributes)

d Subscribe to parameter notifications

e Unsubscribe from parameter notifications

5 Disconnect from server

Data subscription processes

Topics

Request server information

Before you connect to the server running the EK80 program, your client application must obtain information about the server's IP address and command port number.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

RequestServerInfo

Send the following **RequestServerInfo** message to the Processor Unit's specific IP address, or broadcast the message to receive information from all the computers on the local area network. The message must be sent to the user datagram protocol (UDP) port number defined as **Local port** on the **As Server** page in the **Installation** dialog box.

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

ServerInfo2

The EK80 Processor Unit will respond to the requesting client application with a message containing information about the EK80 program. The content of the **ServerInfo2** message is as following.

```
struct ServerInfo2
  {
  char Header[4];                // ”SI2\0”
  char ApplicationType[64];
  char ApplicationName[64];      // Name of the current server application (EK80)
  char ApplicationDescription[128]; // Description of the current application
  long ApplicationID;            // ID of the current application
  long CommandPort;              // Port number to send commands to
  long Mode;             // If the server application (EK80)
       // is running  against the local data source or a remote data source
  char HostName[64];                  // IP address of the Processing Unit the server
       // application is running on
  };
```

CommandPort

The EK80 Processor Unit's UDP port number `CommandPort` must be used from now on to send commands to the EK80 program.

Related topics

Data subscription processes, page 13

Connect to server

Before commands can be sent to the EK80 program, your client application must identify itself to the Processing Unit server application by sending the **ConnectRequest** message. This message must contain user account and password information.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

ConnectRequest

The contents of the **ConnectRequest** message is shown below.

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

Response

The EK80 server application will respond with a **Response** message. If the `ConnectRequest` command succeeded, it will contain at least the client identification. If the `ConnectRequest` command failed, it will contain an error message.

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

The `MsgResponse` field consist of a `ResultCode` and `Parameters`.

1 The `ResultCode` contains the result of the `ConnectRequest` command. The following values are defined.

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

2 The `Parameters` is a comma separated list of various "name:value" pairs that may be present. These parameters are only provided if the **ConnectRequest** was successful. The following values are defined.

- `ClientID`: This is the identification of the current client. The information is used in all further communication with the server application.
- `AccessLevel`: This is the general access level for the current client.

A successful connection will for example provide a **MsgResponse** message.

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

In case the `ConnectRequest` command fails, the `ResponseField` will contain a `ResultInfo` field. This field will contain text describing the failure.

Related topics

Data subscription processes, page 13

Keep connection alive

Once the client application on the local computer is connected to the server application on the EK80 Processor Unit, a two-way monitoring of the communication status must be started. Both the client and the server applications must send the **AliveReport** message once every second.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

AliveReport

The content of the **AliveReport** message is shown using C-code.

```
struct AliveReport
 {
  char Header[4]; // ”ALI\0”
  char Info[1024];
  //The Info field
may for example contain text on the following format: ClientID:1,SeqNo:1\0
 };
```

SeqNo

The `SeqNo` part of the `Info` field shall contain the sequence number of the next request message from the client application to the EK80 server. The sequence number shall start on 1. The client application uses this information to detect if any messages have been lost. If data loss is detected, a re-transmit request issued.

The **AliveReportDef** message from the EK80 to the client application may for example contain:

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

Similar, the `SeqNo` part of the `Info` field shall contain the sequence number of the next response message from the EK80 server to the client. The server application uses this information to detect if any messages have been lost. If data loss is detected, a re-transmit request issued.

Related topics

Data subscription processes, page 13

Issue commands to the server

Once the computer with the client application is connected to the EK80 Processing Unit with the server application (the EK80 program), specific messages are used to issue commands.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

Request

A **Request** message must be sent to the server application (the EK80 program) in order to issue a command on one of the available command targets. This is an example of the main elements of a command request.

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

The `MsgControl` field is formatted as follows: `Sequence number, Current message, Total message`.

1 The `Sequence number` increments for each request message sent to the server.

2 The `Current message` contains the current message in case a request must be split into several UDP messages.

3 The `Total message` contains the total number of messages the current request consists of.

For example, the contents of the `MsgControl` is "`2,1,3`". This means that the current UDP message is the second request message sent from the client to the server, and that the current message is message number 1 of a request that consists of a total of three UDP messages.

The contents of the `MsgRequest` field depend on the current command target. The format is XML based. It must specify a command target, a method on the command target, and any input parameters relevant for the current method.

This is the general structure.

```
<request>
    <clientInfo>
        <cid>clientid</cid>
        <rid>requestid</rid>
    </clientInfo>
    <type>invokeMethod</type>
```

```
    <targetComponent>xx</targetComponent>
    <method>
        <yy>
            <zz></zz>
        </yy>
    </method>
</request>
```

- `clientid`: This is the client identification.
- `requestid`: This is the request identification.
- `xx`: This is the name of the current command target.
- `yy`: This is the name of the current method.
- `zz`: This identifies any parameters of the current method

Response

The server application (the EK80 program) will respond with a **Response** message.

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

The `MsgControl` field is formatted as follows: `Sequence number, Current message, Total message`.

The contents of the **Response** message depend on the current command target. The format is XML based. It contains the result, any error messages, and any output parameters relevant for the current method.

This is the general structure.

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

```
    </yyResponse>
</response>
```

- `clientid`: This is the client identification.
- `requestid`: This is the request identification.
- `error code`: This is the result of the operation. (`0` = OK)
- `yy`: This is the name of the current method.
- `zz`: This identifies any parameters of the current method

Related topics

Data subscription processes, page 13

Collecting data

In order to collect echo data from the server application (the EK80 program), commands must be sent to the **RemoteDataServer** component of the server application.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

The following commands are available.

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

For all commands a **Request** message must be sent from the client application on the local computer to the Processing Unit. A **Response** message is returned from the Processing Unit to the client application. In the following sections only the contents of the `Request` and `Response` fields in the two messages are described.

Topics

Create data subscription, page 20

Handling data, page 21

Change data subscription, page 21

Destroy data subscription, page 22

Create data subscription

The method part of the request shall be set to `<Subscribe>`.

This method has the following input parameters.

- `<requestedPort>`: This is the local area network (LAN) port on the client application computer that shall be used to receive the information.
- `dataRequest`: This is the actual specification of the subscription.

Note

It is not possible to use different ports for each data subscription. All data will be sent to the port specified in the first subscription that is requested.

This is an example of the contents of the `<Request>` field.

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

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

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

This is an example of the contents of the `<Response>` field.

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

```
        </detail>
    </fault>
    <SubscribeResponse>
        <subscriptionID dt="3"></subscriptionID>
    </SubscribeResponse>
</response>
```

Related topics

Data subscription processes, page 13
Collecting data, page 19

Handling data

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

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

 };
```

Related topics

Data subscription processes, page 13
Collecting data, page 19

Change data subscription

The method part of the request shall be set to `ChangeSubscription`.

This method has the following input parameters.

- `subscriptionID`: This is the subscription that shall be changed.
- `dataRequest`: This is the actual specification of the subscription. For more information, see: *Data subscription types*.

This is an example of the contents of the `<Request>` field.

```
<request>
    <ClientInfo>
        <cid>1</cid>
        <rid>1</rid>
    </clientInfo>
    <type>invokeMethod</type>
```

```
    <targetComponent>RemoteDataServer</targetComponent>
    <method>
        <ChangeSubscription>
            <subscriptionID>1</subscriptionID>
            <dataRequest>BottomDetection</dataRequest>
        </ChangeSubscription>
    </method>
</request>
```

The server application will respond with a **Response** message. This method has no output parameters. This is an example of the contents of the `<Response>` field.

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

Related topics

Data subscription processes, page 13
Collecting data, page 19

Destroy data subscription

The method part of the request shall be set to `Unsubscribe`.

This method has the following input parameters.

- **`subscriptionID`**: This is the identification of the subscription that shall be closed.

This is an example of the contents of the `<Request>` field.

```
<request>
    <ClientInfo>
        <cid>1</cid>
        <rid>1</rid>
    </clientInfo>
    <type>invokeMethod</type>
    <targetComponent>RemoteDataServer</targetComponent>
```

```
    <method>
        <Unsubscribe>
            <subscriptionID>1</subscriptionID>
        </Unsubscribe>
    <method>
</request>
```

The server application will respond with a **Response** message. This method has no output parameters. This is an example of the contents of the `<Response>` field.

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

Related topics

Data subscription processes, page 13
Collecting data, page 19

Disconnect from server

The client application on the local computer must send a **DisconnectRequestDef** message to the server application (the EK80 program) when it has finished its operations.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

This method has the following parameters.

Parameter name	Description
<Header>	DIS\0
<szClientInfo>	Name:Simrad;Password:\0

Related topics

Data subscription processes, page 13

Parameter management

Topics

About parameter management

In order to set or get parameters to or from the server application (the EK80 program), certain commands must be sent to the **ParameterServer** component in the application.

Note

In this context the EK80 Processor Unit is regarded as the "server". The EK80 program is the "server application". The program you make yourself, for running on a local computer is referred to as the "client application".

The following methods and commands are available.

- Get parameter
- Set parameter
- Get parameter information (attributes)
- Subscribe to parameter notifications
- Unsubscribe from parameter notifications

The response messages from the **ParameterServer** component contain one additional field.

```
<response>
    <ClientInfo>
        <cid dt="3">clientid</cid>
        <rid dt="3">requestid</rid>
    </clientInfo>
    <fault>
            <detail>
            <message dt="8"></message>
            <errorcode dt="3">0</errorcode>
            <errorcode1 dt="3">0</errorcode1>
```

```
            </detail>
    </fault>
    <yyResponse>
        <zz dt="3"></zz>
    </yyResponse>
</response>
```

`<errorcode1>` is an enhanced result code. The following values are defined.

Error code	Value
PSR_OK	0
PSR_UNSPECIFIED_ERROR	-1
PSR_INVALID_ARGUMENT_ERROR	-2
PSR_EXCEPTION_ERROR	-3
PSR_ACCESS_DENIED_ERROR	-4
PSR_PARAMETER_NOT_FOUND_ERROR	-5
PSR_PARAMETER_ATTRIBUTE_NOT_FOUND_ERROR	-6
PSR_PARAMETER_PROVIDER_NOT_FOUND_ERROR	-7
PSR_VALUE_DATA_TYPE_MISMATCH_ERROR	-8
PSR_VALUE_OUT_OF_RANGE_ERROR	-9
PSR_ARRAY_INDEX_OUT_OF_RANGE_ERROR	-10
PSR_COMMUNICATION_ERROR	-11
PSR_PASSED_LAST_ELEMENT	1
PSR_OPERATION_FORWARDED_TO_REMOTE	2
PSR_HANDLED_BY_OVERRIDE	3

A negative error code indicates that the operation has failed.

Related topics

Parameter management, page 25

Get parameter

The method part of the request shall be set to `<GetParameter>`.

This method has the following input parameters.

- `<ParamName>`: This is the full name of the parameter.
- `<Time>`: This is the time when the value shall be read. It is only available for some parameters. Use 0 (zero) if the latest value is requested.

This is an example of the contents of the `<Request>` field.

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

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

- `<ParamName>`: This is the full name of the parameter.
- `<Time>`: This is the time when the value was updated.

This is an example of the contents of the `<Response>` field.

```
<response>
     <ClientInfo>
          <cid dt="3">1</cid>
          <rid dt="3">28</rid>
     </clientInfo>
     <fault>
          <detail>
               <errorcode dt="3">0</errorcode>
               <errorcode1 dt="3">0</errorcode1>
               <message dt="8"></message>
          </detail>
     </fault>
     <GetParameterResponse>
          <paramValue>
               <value>60</value>
               <time>0</time>
          </paramValue>
     </GetParameterResponse>
</response>
```

Related topics
Parameter management, page 25

Set parameter

The method part of the request shall be set to `<SetParameter>`.

This method has the following input parameters.

- `<ParamName>`: This is the full name of the parameter.
- `<paramValue>`: This is the new value that will be used to update the parameter.
- `<paramType>`: The data variant type of the `<paramValue>` field.

This is an example of the contents of the `<Request>` field. The ping interval is set to 1000 ms, and the data variant type is VT_I4 (4-byte integer).

```
<request>
     <ClientInfo>
          <cid>1</cid>
          <rid>28</rid>
     </clientInfo>
     <type>invokeMethod</type>
     <targetComponent>ParameterServer</targetComponent>


     <method>
          <SetParameter>
               <paramName>
               AcousticDeviceSynchroniser/Interval
               </paramName>
               <paramValue>1000</paramValue>
               <paramType>3</paramType>
          </SetParameter>
     <method>
</request>
```

The server application will respond with a **Response** message. This method has no output parameters. This is an example of the contents of the `<Response>` field.

```
<response>
     <ClientInfo>
          <cid dt="3">1</cid>
          <rid dt="3">28</rid>
     </clientInfo>
     <fault>
          <detail>
```

```
            <errorcode dt="3">0</errorcode>
            <errorcode1 dt="3">0</errorcode1>
            <message dt="8"></message>
        </detail>
    </fault>
</response>
```

Related topics

Parameter management, page 25

Get parameter information (attributes)

The method part of the request shall be set to `<GetParamInfo>`.

This method has the following input parameters.

- `<ParamName>`: This is the full name of the parameter.

This is an example of the contents of the `<Request>` field.

```
<request>
    <ClientInfo>
        <cid>1</cid>
        <rid>28</rid>
    </clientInfo>
    <type>invokeMethod</type>
    <targetComponent>ParameterServer</targetComponent>
    <method>
        <GetParamInfo>
            <paramName>
            AcousticDeviceSynchroniser/Interval
            </paramName>
        </GetParamInfo>
    <method>
</request>
```

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

- `<Attributes>`: This is the attributes of the current parameter.
- `<Value>`: This is the value of the parameter.
- `<TimeStamp>`: This is the time when the value was updated.

This is an example of the contents of the `<Response>` field.

```
<response>
    <ClientInfo>
```

```
        <cid dt="3">1</cid>
        <rid dt="3">28</rid>
    </clientInfo>
    <fault>
        <detail>
            <errorcode dt="3">0</errorcode>
            <errorcode1 dt="3">0</errorcode1>
            <message dt="8"></message>
        </detail>
    </fault>
    <GetParamInfoResponse>
        <paramInfo>
            <attributes>
                <AccessLevel dt="3">0</AccessLevel>
                <Default dt="3">0</Default>
                <Description dt="8">
                Setting the ping interval in ms
                </Description>
                <Maximum dt="3">2147483647</Maximum>
                <Minimum dt="3" >-2147483648</Minimum>
                <Name dt="8">Interval</Name>
                <Status dt="3">0</Status>
                <Type dt="3">3</Type>
            </attributes>
            <value dt="3">1000</value>
            <timeStamp dt="8">0</timeStamp>
        </paramInfo>
    </GetParameInfoResponse>
</response>
```

Related topics

Parameter management, page 25

Subscribe to parameter notifications

The method part of the request shall be set to `<Subscribe>`.

This command can be used to enable parameter notification for one single parameter. The command must be repeated for each individual parameter if notifications are required for multiple parameters. This method has the following input parameters.

- `<ParamName>`: This is the full name of the parameter.
- `<requestedPort>`: This is the local area network (LAN) port on the client application computer that shall be used to receive the information.

This is an example of the contents of the <Request> field.

```
<request>
     <ClientInfo>
          <cid>1</cid>
          <rid>29</rid>
     </clientInfo>
     <type>invokeMethod</type>
     <targetComponent>ParameterServer</targetComponent>
     <method>
          <Subscribe>
          <paramName>AcousticDeviceSynchroniser/Interval</param-
          Name>
          <requestedPort>32145</requestedPort>
          </Subscribe>
     <method>
</request>
```

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

- <cookie>: This is the identification of the current subscription.

This is an example of the contents of the <Response> field.

```
<response>
     <ClientInfo>
          <cid dt="3">1</cid>
          <rid dt="3">29</rid>
     </clientInfo>
     <fault>
          <detail>
               <errorcode dt="3">0</errorcode>
               <errorcode1 dt="3">0</errorcode1>
               <message dt="8"></message>
          </detail>
     </fault>
     <SubscribeResponse>
          <cookie>10001</cookie>
     </SubscribeResponse>
</response>
```

Related topics

Parameter management, page 25

Unsubscribe from parameter notifications

The method part of the request shall be set to `<CloseSubscription>`.

This command can be used to stop parameter notification for one single parameter. The command must be repeated for each individual parameter if notifications from multiple parameters shall be stopped. This method has the following input parameters.

- `<ParamName>`: This is the full name of the parameter.
- `<cookie>`: This is the identification of the current subscription.

This is an example of the contents of the `<Request>` field.

```
<request>
     <ClientInfo>
          <cid>1</cid>
          <rid>29</rid>
     </clientInfo>
     <type>invokeMethod</type>
     <targetComponent>ParameterServer</targetComponent>
     <method>
          <CloseSubscription>
               <paramName>
               AcousticDeviceSynchroniser/Interval
               </paramName>
               <cookie>10001</cookie>
          </CloseSubscription>
     <method>
</request>
```

The server application will respond with a **Response** message. Except for the error codes, the `<CloseSubscription>` method has no output parameters. This is an example of the contents of the `<Response>` field.

```
<response>
     <ClientInfo>
          <cid dt="3">1</cid>
          <rid dt="3">29</rid>
     </clientInfo>
     <fault>
          <detail>
               <errorcode dt="3">0</errorcode>
               <errorcode1 dt="3">0</errorcode1>
               <message dt="8"></message>
          </detail>
     </fault>
</response>
```

Related topics

Parameter management, page 25

Handling parameter subscriptions

Topics

About subscriptions handling

Parameter notifications are sent from the server application (the EK80 program) to the client through a separate UDP connection where the data are packed in dynamic binary structures. This ensures an efficient distribution of parameter notifications.

The communication protocol used for parameter notification shall ensure that all notifications are delivered in the same chronological order as they occurred in the server. The following messages are part of the parameter subscription management:

- **Value and attribute update** messages
- **Server alive** message
- **Request re-transmit** message

The basic principles of operation are that the server sends value and attribute update messages periodically when there are parameter value/attribute changes, or a server alive message if no parameter value/attribute has changed. The client can send a request re-transmit message if it detects that is has lost any messages.

Related topics

Handling parameter subscriptions, page 34

The header structure (`ParamMsgHdrDef`)

All LAN messages involved in parameter subscription management starts with a `ParamMsgHdrDef` structure.

```
typedef struct
  {
  unsigned short usCurrentMsg; // This is the current
UDP package number.
  unsigned short usTotalMsg; // This is the total number
```

```
of UDP packages.
  long lSeqNo;
  long lMsgID;
  } ParamMsgHdrDef;
```

`lSeqNo`: This is the “Value and attribute update" number. It is incremented for each UDP package. In a **Server alive** message this means the last used sequence from the server. In an **Request retransmit** message, this means the UDP package that should be re-transmitted.

`lMsgID`: This is the identification of the current message. The following values are defined.

- `const long lPM_SERVER_ALIVE = 3;`
- `const long lPM_REQUEST_RETRANSMIT = 4;`
- `const long lPM_VALUE_ATTRIBUTE_UPDATE = 6;`

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

The actual parameter notification data is organised as follows:

- **Value and attribute update** messages
  - Parameter update structures
  - Attribute update structures
- **Server alive** message
- **Request re-transmit** message

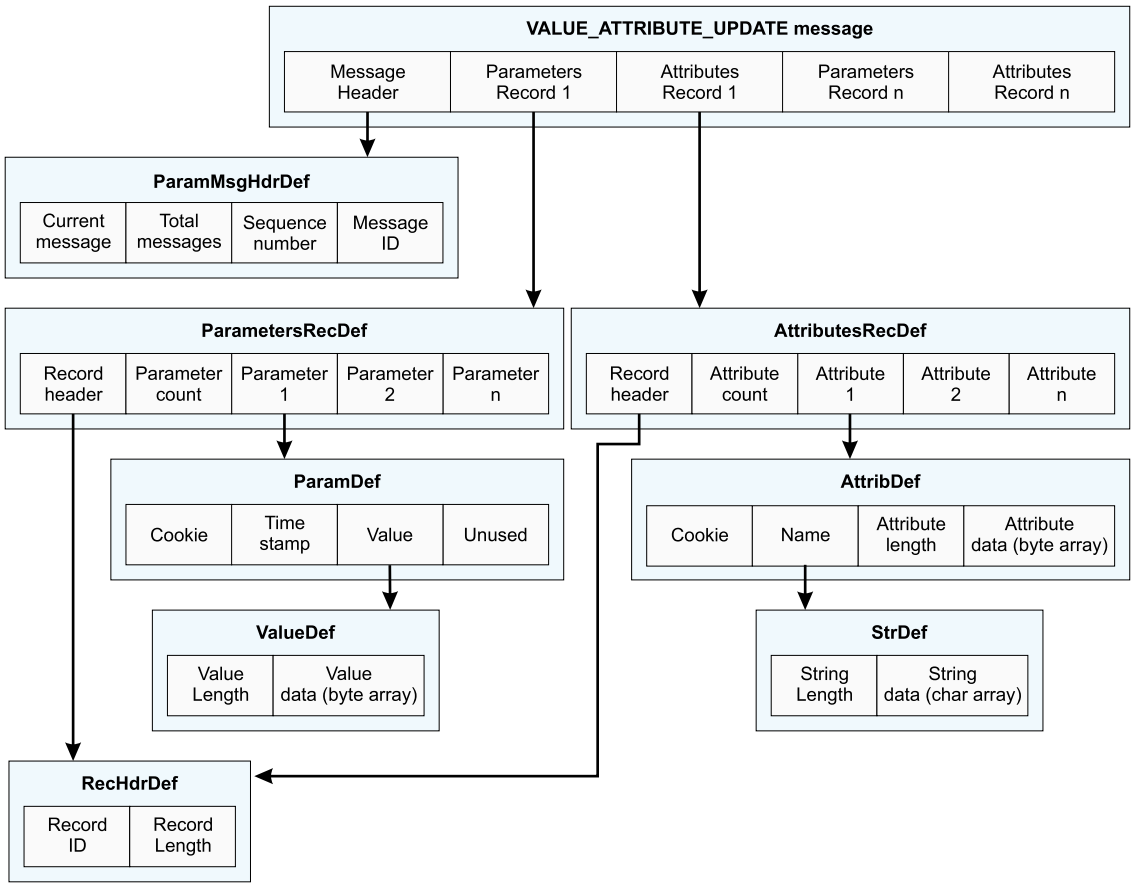
Related topics

Handling parameter subscriptions, page 34

Value and attribute update messages

The content of the **Value and attribute update** message is highly dynamic. It will only contain data for parameter values/attributes that have changed since the last notification message.

The message starts with a `ParamMsgHdrDef` and continues with either a `ParametersRecDef` or an `AttributesRecDef`. The message may actually consists of multiple of these structures interleaved. This is due to the need to preserve the order in which these notifications where generated on the server.



At the "top" level the message contains one of two records (parameter changes or attribute changes). Each record has a record header as follows:

```
typedef struct
  {
  unsigned short usRecID; // This is the identification
of the current record.
  unsigned short usRecLen; // This is the length
(in bytes) of the current record (including this header).
  } RecHdrDef;
```

`usRecID`: This is the identification of the current record. The following values are defined.

- `const unsigned short RID_VALUE_UPDATE = 11;`
- `const unsigned short RID_ATTRIBUTE_UPDATE = 12;`
- `const unsigned short RID_END = 0xdead;`

Topics

Parameter update structures, page 37

Attribute update structures, page 38

Parameter update structures

Three structures are used to convey information about parameter value update notifications.

1

```
typedef struct
  {
  RecHdrDef hdr; // This is the identification
of the current structure. (hdr.usRecID = 11
  long lParamCount; // This is the number
of parameter updates contained in the current structure.
  ParamDef aParams[1]; // This is a dummy placeholder. The size of this field
depends on the number of parameter notifications that is contained
in the message, and the size of each parameter notification.
  } ParametersRecDef;
```

2

```
typedef struct
  {
  long lCookie; // This is the identification
of the current parameter. The value is returned
from the Start parameter notification.
  DWORDLONG dwlTimeStamp; // This is the time when
the parameter was updated.
  ValueDef sValue; This is the actual
parameter data.
  long unused;
  } ParamDef;
```

3

```
typedef struct
  {
 long lValueLen; // This is the size of
the parameter data byte array following below.
  BYTE aValue[1];
  } ValueDef;
```

`aValue[1]`: This is the byte array holding the updated parameter value. The client must be able to match this byte array against the actual data type of the parameter. This can be done either by hard coding with knowledge about the size of the parameter, or by requesting parameter information from the server.

Related topics

Handling parameter subscriptions, page 34
Value and attribute update messages, page 35

Attribute update structures

Three structures are used to convey information about parameter attribute update notifications.

The content of the structures must be decoded by starting at the beginning of the received byte array. Use the size information contained in the various items to advance through the message until all data is processed.

1

```
typedef struct
  {
  RecHdrDef hdr; // Identification of the current structure(hdr.usRecID = 12)
  long lAttribCount; // Number of attribute updates contained in the
current structure   AttribDef aAttribs[1]; // Dummy placeholder.
Size of the field depends on the number         of attribute updates
in the message and the size of each attribute update
  } AttributesRecDef;
```

2

```
typedef struct
  {
  long lCookie; // Identification of the current parameter
  STRDef sAttributeName; // This is the name of
the current attribute.
  long lAttributeLen; // This is the size of
the attribute data byte array following below.
        of attribute updates in the message and the sizee of each attribute update
  BYTE aAttribute[1]; // This is the Byte array
holding the updated attribute value. The client must be able to match
this byte array against the actual data type of the attribute. This can be done
either by hard coding with knowledge about the size of the parameter,
or by requesting parameter information from the server.
  } AttribDef;
```

3

```
typedef struct
  {
  long lStrLen; // This is the size of
the character array following below.
  char aStr[1]; // This is the actual
string data.
  } STRDef;
```

Related topics

Handling parameter subscriptions, page 34
Value and attribute update messages, page 35

Server alive message

This message is sent periodically from the server (the EK80 program) when there are no changes to values or attribute. The message can be used by the client application to monitor the health of the server, and to detect if the client has lost any messages.

```
typedef struct
  {
  ParamMsgHdrDef hdr;
  } AliveMessageDef;
```

- The `lMsgID` field of the `hdr` will have the value 3 (`const long lPM_SERVER_ALIVE = 3`).
- The `lSeqNo` field of the `hdr` contains the sequence number of the last update message sent from the server. If the sequence number is different from what was expected, the client application must send a **Re-transmit** request to the server.

Related topics

Handling parameter subscriptions, page 34

Request re-transmit message

This message can be sent by the client application to the server (the EK80 program) if it detects that messages have been lost.

```
typedef struct
  {
  ParamMsgHdrDef hdr;
  } RequestRetransmitMessageDef;
```

- The `lMsgID` field of the `hdr` will have the value 4 (`const long lPM_REQUEST_RETRANSMITT = 4`).
- The `lSeqNo` field of the `hdr` must be set to the message/UDP package that shall be re-transmitted.

Related topics

Handling parameter subscriptions, page 34

Data subscription types

Topics

About data subscription types

Specific data subscription *types* are used to enable the data subscription and remote control.

Each data subscription requires an identifier to identify from which subscription the data is requested. **ChannelID** (channel identifier) is used for the frequency channel in normal echo sounder operations. A comma separated list of available identifiers can be obtained from the parameter **TransceiverMgr/Channels. VirtualChannelID** is the identifier during missions and ADCP velocity subscriptions. The **VirtualChannelID** comprises a **ChannelID** and a **PingID**. The Ping ID identifies specific pings in a ping sequence from a transducer. The virtual channel name is currently not fully supported by the subscription

functionality. Therefore when making a subscription to a logical channel, data for the first defined virtual channel related to this particular logical channel (transducer) is returned.

The data output always starts with a 64-bit integer time value. This value identifies the number of 100 nanoseconds intervals that has elapsed since January 1, 1601.

The subscription input string is a comma separated list of input parameters. Numerical parameters must use "." (full stop) as decimal separator. If a parameter is skipped in the subscription input string it will be replaced by its default value.

In the output data to be received by external devices, the information about the current structure length is provided. From this information, the number of elements in an array may be calculated.

Note

The decoding of the subscription is case insensitive.

Related topics

Data subscription types, page 40

Data subscription type: ADCP Backscatter

There are four separate backscatter data subscription types, one for each ADCP beam. Both input and output strings are equivalent for all of them. Only the data subscription strings differ.

Input

The data subscription strings are:

- BackscatterFore
- BackscatterAft
- BackscatterStarboard
- BackscatterPort
- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

BackscatterType

Description	Range	Unit	Type	R/O
This is the selection of ADCP beams to subscribe backscatter from.	backscatter-fore backscatter-aft backscatter-port backscatter-starboard		VT_BSTR	N/A
Example:	backscatter-fore			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

UseHeave

Description	Range	Unit	Type	R/O
This is a boolean operator for using or not the heave.	true false		VT_I4	N/A
Example:	true			

ScatterType

Description	Range	Unit	Type	R/O
This is the TVG function.	power sv sp ss tvg-20 tvg-40		VT_BSTR	N/A
Example:	sv			

Output

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

Example 1 BackscatterFore

```
ADCP,
BackscatterFore,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 2 BackscatterStarboard

```
ADCP,
BackscatterStarboard,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 3 BackscatterAft

```
ADCP,
BackscatterAft,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 4 BackscatterPort

```
ADCP,
BackscatterPort,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Related topics

Data subscription types, page 40

Data subscription type: ADCP Beamdata

The ADCP Beamdata subscription provides velocity data for the individual Janus beams, signal correlation level for the individual Janus beams or error velocity. Set **Beam-data-type** to get the selected values in return. The ADCP Beamdata subscription string is **AdcpBeamdata**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

Beam-data-type

Description	Range	Unit	Type	R/O
This is the selection of ADCP beams and data types to subscribe data from.	beam-velocity-1 beam-velocity-2 beam-velocity-3 beam-velocity-4 correlation-beam1 correlation-beam2 correlation-beam3 correlation-beam4 error-factor		VT_BSTR	N/A
Example:	beam-velocity-1			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Use-heave-adjustment

Description	Range	Unit	Type	R/O
This is a boolean operator for applying or not the heave adjustment.	true false		VT_BSTR	N/A
Example:	true			

Output

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

Example 1 beam-velocity-1

```
AdcpBeamData,
beam-velocity-1,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 2 beam-velocity-2

```
AdcpBeamData,
beam-velocity-2 ,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 3 correlation-beam1

```
AdcpBeamData,
correlation-beam1,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 4 correlation-beam2

```
AdcpBeamData,
correlation-beam2,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Related topics

Data subscription types, page 40

Data subscription type: ADCP Bottom Detector

The ADCP Bottom detector subscription type provides recorded depth for each Janus beam. The depth is converted from the slant angle of the beams to true depth. The ADCP Bottom detector subscription string is **AdcpBottomSubscription**. This subscription type is only available from REST API.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default. (CR+LF)

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Use-heave-adjustment

Description	Range	Unit	Type	R/O
This is a boolean operator for applying or not the heave adjustment.	true false		VT_BSTR	N/A
Example:	true			

Output

Subscription to be set up from REST API. For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*.
After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Related topics

Data subscription types, page 40

Data subscription type: ADCP Geo Velocity

There are four separate ADCP geo velocity data subscription types. The geo velocity is relative to the earth coordinates and orientation. Both input and output strings are equivalent for all of them. Only the data subscription strings differ.

Input

The data subscription strings are:

- VelocityNorth
- VelocityEast
- VelocityVertical
- VelocityGeo
- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

CellsNumber

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

StopAtBottom

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Output

```
struct StructVelocity
  {
  StructVelocityHeader VelocityHeader;
  StructVelocityData VelocityData;
  };
```

```
struct StructVelocityHeader
  {
  double range; // (m)
  double rangeStart; // (m)
  double BottomDepth; // (m) This is the detected bottom depth.
  int errorCode; // (dB) Defined error codes
  };
```

```
struct StructVelocityData
  {
  int numValidVelocities;
  StructVelocityVectorHcs Velocities[2500];
  };
```

```
struct StructVelocityVectorHcs
  {
  float x; // (m): Velocity measured in x, y, z direction relative to the vessel.
  float y;
  float z;
  };
```

- `CellsNumber` (): 1–2500, default is 1000
- `StopAtBottom` (): Number identifying bottom detection function, full range (0), stop at first hit (1), stop at last hit (2).

For REST API, use the generic **AdcpVelocity** subscription.

Subscription string example

Example 1 VelocityNorth

```
ADCP,
VelocityNorth,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

Example 2 VelocityGeo

```
ADCP,
VelocityGeo,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
```

```
CellsNumber=2000,
StopAtBottom=1,
```

Example 3 VelocityEast

```
ADCP,
VelocityEast,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=1200,
StopAtBottom=2,
```

```
CellsNumber=2000,
StopAtBottom=1,
```

Example 4 VelocityVertical

```
ADCP,
VelocityVertical,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=1000,
StopAtBottom=1,
```

Related topics

Data subscription types, page 40

Data subscription type: ADCP Output

The purpose of the ADCP Output subscription type is setting up the output of the ZMQ beam velocity data. This subscription type is only available from REST API.

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Decimation

Description	Range	Unit	Type	R/O
This is an optional string.	none 10 cell-size		VT_BSTR	N/A
Example:	10			
For future use. Currently, the velocity output is always decimated by 10.				

Output

Output enabled from GUI or REST API.

Related topics

Data subscription types, page 40

Data subscription type: ADCP Quality Factor

The ADCP Quality factor subscription provides the Percent Good number for the given ADCP Editing setting. The ADCP Quality factor subscription string is **AdcpQualityFactor**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Stop-at-bottom

Description	Range	Unit	Type	R/O
This is used to control the end range of the velocity output.	full first-hit last-hit		VT_BSTR	N/A
Example:	full			
This is an optional string.				

Quality-factor-average

Description	Range	Unit	Type	R/O
This is a boolean operator. Select between single ping quality factor numbers or numbers based on the current averaging of current velocity output.	true false		VT_I4	N/A
Example:	false			

Use-heave-adjustment

Description	Range	Unit	Type	R/O
This is a boolean operator for applying or not the heave adjustment.	true false		VT_BSTR	N/A
Example:	true			

Output

Subscription to be set up from REST API. For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*.
After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Related topics

Data subscription types, page 40

Data subscription type: ADCP Velocity

The ADCP Velocity subscription string is **AdcpVelocity**. The vessel velocity is relative to the vessel orientation. The geo velocity is relative to the earth coordinates. Both input and output strings are equivalent for all of them. Only the data subscription strings differ.

Input

The data subscription strings are:

- VelocityFore
- VelocityStarboard
- VelocityDown
- VelocityVessel

- VelocityNorth
- VelocityEast
- VelocityVertical
- VelocityGeo
- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

VelocityType

Description	Range	Unit	Type	R/O
This is the velocity type.	velocity-fore velocity-starboard velocity-down velocity-vessel velocity-north velocity-east velocity-vertical velocity-geo		VT_BSTR	N/A
Example:	velocity-fore			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Average

Description	Range	Unit	Type	R/O
This is the number of pings to be averaged for the velocity output.	<0, 300>		VT_I4	N/A
Example:	1			

Stop-at-bottom

Description	Range	Unit	Type	R/O
This is used to control the end range of the velocity output.	full first-hit last-hit		VT_BSTR	N/A
Example:	full			
This is an optional string.				

Use-heave-adjustment

Description	Range	Unit	Type	R/O
This is a boolean operator for applying or not the heave adjustment.	true false		VT_BSTR	N/A
Example:	true			

Decimation

Description	Range	Unit	Type	R/O
This is an optional string.	none 10 cell-size		VT_BSTR	N/A
Example:	10			
For future use. Currently, the velocity output is always decimated by 10.				

Output

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

Example 1 VelocityNorth

```
ADCP,
VelocityNorth,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Example 2 VelocityFore

```
ADCP,
VelocityFore,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Related topics

Data subscription types, page 40

Data subscription type: ADCP Vessel Velocity

There are four separate ADCP vessel velocity data subscription types. The vessel velocity is relative to the vessel orientation and velocity. Both input and output strings are equivalent for all of them. Only the data subscription strings differ.

Input

The data subscription strings are:

- VelocityFore
- VelocityStarboard
- VelocityDown
- VelocityVessel
- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

CellsNumber

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

StopAtBottom

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Output

```
struct StructVelocity
  {
  StructVelocityHeader VelocityHeader;
  StructVelocityData VelocityData;
  };
```

```
struct StructVelocityHeader
  {
  double range;
  double rangeStart;
  double BottomDepth; // (m) This is the detected bottom depth.
  int errorCode; // (dB) Defined error codes
  };
```

```
struct StructVelocityData
  {
  int numValidVelocities;
  StructVelocityVectorHcs Velocities[2500];
  };
```

```
struct StructVelocityVectorHcs
  {
  float x; // (m) Velocity measured in x, y, z direction relative to the vessel.
  float y;
  float z;
  };
```

For REST API, use the generic **AdcpVelocity** subscription.

Subscription string example

Example 1 VelocityFore

```
ADCP,
VelocityFore,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Example 2 VelocityVessel

```
ADCP,
VelocityVessel,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Example 3 VelocityStarboard

```
ADCP,
VelocityStarboard,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Example 4 VelocityDown

```
ADCP,
VelocityDown,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
CellsNumber=2000,
StopAtBottom=1,
```

Related topics

Data subscription types, page 40

Data subscription type: Bottom Detection

The bottom detection subscription provides detected bottom depth and bottom reflectivity for the selected channel. The bottom detection data subscription string is **BottomDetection**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

UpperDetectorLimit

Description	Range	Unit	Type	R/O
This is the **Minimum Depth** setting used for bottom detection.	<0,10000>	m	VT_R8	N/A
Example:	10			

LowerDetectorLimit

Description	Range	Unit	Type	R/O
This is the **Maximum Depth** setting used for bottom detection.	<0,10000>	m	VT_R8	N/A
Example:	1000			

BottomBackstep

Description	Range	Unit	Type	R/O
This is the **Bottom Backstep** setting for the channel.	<-100,10>	dB	VT_R8	N/A
Example:	-50			

Output

```
struct StructBottomDepthHeader
  {
  DWORDLONG dlTime;
  };
struct StructBottomDepthData
  {
  double dBottomDepth; // (m) This is the detected bottom depth.
  double dReflectivity; // (dB) This is the seabed reflectivity.
    If
the transducer faces upwards, use the surface reflectivity.
  double dVesselLogDistance; //  (nmi) This is the vessel's
```

```
sailed distance.
  };
struct StructBottomDepth
  {
  StructBottomDepthHeader BottomDepthHeader;
  StructBottomDepthData BottomDepthData;
  };
```

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

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

Related topics

Data subscription types, page 40

Data subscription type: Echogram

The echogram subscription provides data for drawing the echogram. Selection of the echogram type and the update rate is available. The echogram data subscription string is .**Echogram**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

PixelCount

Description	Range	Unit	Type	R/O
This is the number of pixels/values to be returned by this subscription.	<0,10000>		VT_I4	N/A
Example:	500			
The compression and the expansion depend on whether there are more or less samples than the pixel count. The pixel value is given as 16-bit EqInt in decibels.				

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

TVGType

Description	Range	Unit	Type	R/O
This is the Time Variable Gain (TVG) parameter setting.	Pr Sv Sp TS SpAndTS		VT_BSTR	N/A
Example:	Sv			

EchogramType

Description	Range	Unit	Type	R/O
This is the echogram type in use.	Surface Bottom Trawl		VT_BSTR	N/A
Example:	Surface			
It sets the direction of the range: towards the bottom for **SurfaceType** and towards the surface for **BottomType**. Depending on the echogram type, the range and the start range are established relative to surface, bottom or trawl (towed vehicle).				

CompressionType

Description	Range	Unit	Type	R/O
This is the interpolation method. If the number of samples is larger than the requested number of pixels, select how to interpolate.	Mean Peak		VT_BSTR	N/A
Example:	Mean			

ExpansionType

Description	Range	Unit	Type	R/O
This is the extrapolation method. If the number of samples is less than the requested number of pixels, select how to extrapolate.	Interpolation Copy		VT_BSTR	N/A
Example:	Interpolation			

EchogramTemporalAverage

Description	Range	Unit	Type	R/O
This is the time between two pings (echogram data transfers). All data will be averaged during those two pings.	<-2, 20>	s	VT_I4	N/A
Example:	-1			
This defines the number of seconds to be averaged. The ping rate determines the number of pings that are averaged.				

EchogramHeave

Description	Range	Unit	Type	R/O
This is a boolean operator (0 or 1) for compensating or not the heave.	<0, 1>		VT_I4	N/A
Example:	1			
Use 1 to have heave compensated pixels.				

EchogramTransducerDepth

Description	Range	Unit	Type	R/O
This is a boolean operator (0 or 1) for compensating or not the transducer depth.	<0, 1>		VT_I4	N/A
Example:	1			
This makes range relative to the transducer (0) or to the surface (1).				

EchogramMinPixelValue

Description	Range	Unit	Type	R/O
This is a threshold that allows to ignore values before compensating or expanding.	<-200, 100>		VT_I4	N/A
Example:	-100			
Suppress weak echoes by limiting the decibels. Values below threshold are set to -20000.				

EchogramPingFilterState

Description	Range	Unit	Type	R/O
This is the **Ping-Ping Filter** function.	<0, 3>		VT_I4	N/A
Example:	0			
Normally not used by EK80 Used with the **Noise Reduction** function set to *Off*, *2 of 3*, *2 of 2* or *3 of 3*				

EchogramStopAtBottom

Description	Range	Unit	Type	R/O
This is a boolean operator (0 or 1) for truncating or not the data array if the depth is below the bottom.	true false		VT_I4	N/A
Example:	false			
This will truncate the data array at the bottom minus the **EchogramBottomMargin**.				

EchogramBottomMargin

Description	Range	Unit	Type	R/O
This is used to limit the amount of data to be transferred.	<-10, 20>	m	VT_I4	N/A
Example:	0.5			
See: ***EchogramStopAtBottom*** If you set **EchogramBottomMargin** to 1 metre and **EchogramStopAtBottom** to 1 metre, the EK80 system will provide pixels down to 1 metre above the detected bottom.				

EchogramDelay

Description	Range	Unit	Type	R/O
This is used by systems with multiple transmit groups to identify the transmit pulse corresponding to the selected beam.				N/A
Example:				
Not used by EK80 This is only used by ME70/MS70 with pulses transmitted in multiple groups with increasing delay.				

TVGFunction

Description	Range	Unit	Type	R/O
This is the Time Variable Gain (TVG) parameter setting.	<0, 100>		VT_I4	N/A
Example:	20			
It only works when **TVGType** or **BottomTVGType** are set to *user*.				

BottomGain

Description	Range	Unit	Type	R/O
It allows you to amplify the echo by the selected number of decibels (dB) from the detected bottom and down.	<-100, 100>		VT_I4	N/A
Example:	0			
Normally not used by EK80.				

BottomTVGType

Description	Range	Unit	Type	R/O
This is the Bottom Time Variable Gain (TVG) parameter setting.	none sv ts user		VT_BSTR	N/A
Example:	none			
Normally not used by EK80 Select *None* to keep the TVG parameter as selected in **TVGType**. Select *Sv, Ts, User* to change the TVG from the detected bottom and down.				

Output

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

- `nEchogramElement[30000]`: This is the echogram information on a 16-bit logarithmic format.

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
Echogram,
ChannelID=<ChannelID>,
PixelCount=500,
Range=200,
RangeStart=3,
TVGType=TS,
EchogramType=Surface,
CompressionType=Mean,
ExpansionType=Interpolation
```

Related topics

Data subscription types, page 40

Data subscription type: Integration

The integration data subscription string is **Integration**.

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

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

LayerType

Description	Range	Unit	Type	R/O
This is the current layer type.	Surface Bottom Pelagic		VT_BSTR	N/A
Example:	Surface			

IntegrationState

Description	Range	Unit	Type	R/O
This is the "on/off" switch for the integration.	Start Stop		VT_BSTR	N/A
Example:	Start			

Pulse form

Description	Range	Unit	Type	R/O
This option is used for pulse form. The different types of pulse forms will be induce different calculations of integration	CW Chirp		VT_BSTR	N/A
Example:	CW			

Update

Description	Range	Unit	Type	R/O
This is a "selector" that allows you to provide integration value for latest ping or for the calculation interval.	UpdatePing UpdateAccumulate		VT_BSTR	N/A
Example:	UpdatePing			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Margin

Description	Range	Unit	Type	R/O
This is a range selector. Use it to set the range over the detected bottom. This range will not be included in integration.	<0,200>	m	VT_I4	N/A
Example:	1			

SvThreshold

Description	Range	Unit	Type	R/O
This is a threshold selector. Set this threshold to remove the weakest echoes from the integration.	<-200,100>	dB	VT_R8	N/A
Example:	-100			

Output

```
struct StructIntegrationDataHeader
  {
  DWORDLONG dlTime;
  };
struct StructIntegrationDataBody
  {
  double dSa;
  };
struct StructIntegrationData
  {
  StructIntegrationDataHeader IntegrationDataHeader;
  StructIntegrationDataBody IntegrationDataBody;
  };
```

- `dSa` (m^2/nmi^2): In this context the $\mathbf{s_A}$ value presents the current biomass index.

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

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

Related topics

Data subscription types, page 40

Data subscription type: Integration chirp

The integration chirp data subscription string is **IntegrationChirp**.

Note

This data subscription type is deprecated. Use the data subscription type **Integration** *and the parameter "Pulse Form" to indicate what type of pulse form is in use.*

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

LayerType

Description	Range	Unit	Type	R/O
This is the current layer type.	Surface Bottom Pelagic		VT_BSTR	N/A
Example:	Surface			

IntegrationState

Description	Range	Unit	Type	R/O
This is the "on/off" switch for the integration.	Start Stop		VT_BSTR	N/A
Example:	Start			

Update

Description	Range	Unit	Type	R/O
This is a "selector" that allows you to provide integration value for latest ping or for the calculation interval.	UpdatePing UpdateAccumulate		VT_BSTR	N/A
Example:	UpdatePing			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Margin

Description	Range	Unit	Type	R/O
This is a range selector. Use it to set the range over the detected bottom. This range will not be included in integration.	<0,200>	m	VT_I4	N/A
Example:	1			

SvThreshold

Description	Range	Unit	Type	R/O
This is a threshold selector. Set this threshold to remove the weakest echoes from the integration.	<-200,100>	dB	VT_R8	N/A
Example:	-100			

SvBinOverlap

Description	Range	Unit	Type	R/O
This parameter controls how much layer segments overlap.	<0,100>	%	VT_I4	N/A
Example:	50			
This is a parameter from the **Layer Properties** dialog box. When the EK80 calculates the volume backscatter, it divides the total vertical range of the depth layer to "segments". These segments are stacked on top of each other. The segments may overlap, and the **Sv Bin Overlap** option controls (in %) how much they overlap.				

Output

```
struct StructIntegrationDataHeader
  {
  DWORDLONG dlTime;
  double dSa;
  double Distance;
  float frequencyLimits[2];
  float SVFrequencies[1000];
  };
```

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
Integration,
ChannelID=<ChannelID>,
State=Start,
Update=UpdatePing,
Layertype=Surface,
Range=100,
Rangestart=10,
Margin=0.5,
SvThreshold=-100.0,
SvBinOverlap=50
```

Related topics

Data subscription types, page 40

Data subscription type: Noise Spectrum

The noise spectrum subscription provides the noise levels for the frequency band of the selected channel. The noise spectrum data subscription string is **NoiseSpectrum**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

State

Description	Range	Unit	Type	R/O
This is the "on/off" switch for the noise measurements.	Start Stop	m	VT_R8	No
Example:	Start			

Output

```
struct StructNoiseSpectrumData
  {
  DWORDLONG dlTime;
  float NoiseLevel[1000];
  };
```

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
NoiseSpectrum,
ChannelID=<ChannelID>,
Range=100,
Rangestart=10,
State=Start
```

Related topics

Data subscription types, page 40

Data subscription type: Sample data

The sample data subscription provides full sample range on the requested **SampleData** type format. The sample data subscription string is **SampleData**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

SampleDataType

Description	Range	Unit	Type	R/O
This is the type of sample data requested for this subscription.	Power Angle Sv Sp Ss TVG20 TVG40 PowerAngle Complex4Float32 Complex3 Float32		VT_BSTR	N/A
Example:	Power			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

AutoRange

Description	Range	Unit	Type	R/O
This is a boolean operator (0 or 1) which is used as an alternative to set fixed range.	<0, 1>		VT_I4	N/A
Example:	0			
This will transmit all samples collected for the ping, overriding range.				

SampleDataHeave

Description	Range	Unit	Type	R/O
Set to 0 to turn off heave compensation of data.	<0, 1>		VT_I4	N/A
Example:	1			
It only works when **Sample Transducer Depth** is set to 1.				

SampleTransducerDepth

Description	Range	Unit	Type	R/O
Set to 1 to have samples relative to the sea surface.	<0, 1>		VT_14	N/A
Example:	1			

SampleDataDelay

Description	Range	Unit	Type	R/O
This is used by systems with multiple transmit groups to identify the transmit pulse corresponding to the selected beam.	<0, 1>		VT_I4	N/A
Example:	1			
Not relevant for echo sounders.				

ADCPSubFeature

Description	Range	Unit	Type	R/O
Use it when the subscription is for data from an ADCP beam.	fore aft starboard port		VT_BSTR	N/A
Example:	fore			

Function

Description	Range	Unit	Type	R/O
This is used to select the data source for samples.	real-part amplitude		VT_BSTR	N/A
Example:	amplitude			
Not relevant for echo sounders.				

Output (Power, Angle, Sv, Sp, Ss, TVG20, TVG40)

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

- `nSampleDataElement[30000]`: This is a 16-bit sample in logarithmic format.

Output (PowerAngle)

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

- `nSampleDataElement[60000]`: This is a composite sample array for power and angle.
- `nPowerValues`: This is the number of power samples.
- `nAngleValues`: This is the number of angle values.

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
SampleData,
ChannelID=<ChannelID>,
SampleDataType=Power,
Range=200,
RangeStart=3,
```

Related topics

Data subscription types, page 40

Data subscription type: System State

The system state subscription provides an alive message with information about the position, ping activity, etc.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

Update-period

Description	Range	Unit	Type	R/O
This is the update period.	<100, 10000>	s	VT_I4	N/A
Example:	1000			

Output

Subscription to be set up from REST API. For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*.
After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Related topics

Data subscription types, page 40

Data subscription type: Target Strength (TS) detection

The target strength (TS) detection subscription provides single targets detected from a CW pulse. The target strength (TS) detection data subscription string is **TSDetection**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

LayerType

Description	Range	Unit	Type	R/O
This is the current layer type.	Surface Bottom Pelagic		VT_BSTR	N/A
Example:	Surface			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

MinTSValue

Description	Range	Unit	Type	R/O
This is the Minimum Target Strength (TS) Value, also known as Minimum Threshold.	<-120,50>	dB	VT_R8	N/A
Example:	–50.0			
This is a parameter setting in **Single Target Detection** dialog box. The target strength (TS) for a single target must exceed this threshold (in dB) to be accepted. This setting applies to both CW and FM operation.				

MinEchoLength

Description	Range	Unit	Type	R/O
This is the Minimum Echo Length,	<0,20>		VT_R8	N/A
Example:	0.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Minimum Echo Length** to 0.8, all echoes shorter than 0.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxEchoLength

Description	Range	Unit	Type	R/O
This is the Maximum Echo Length.	<0,20>		VT_R8	N/A
Example:	1.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Maximum Echo Length** to 1.8, all echoes longer than 1.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxGainCompensation

Description	Range	Unit	Type	R/O
This is the Maximum Gain Compensation.	<0,12>		VT_R8	N/A
Example:	6.0			
This is a parameter setting in **Single Target Detection** dialog box. Targets located off centre will offer weaker echoes due to the beam properties. The EK80 system automatically compensates for this using a mathematical model, and you can manually control the effect of this algorithm by defining a maximum gain value. This setting applies to both CW and FM operation.				

MaxPhaseDeviation

Description	Range	Unit	Type	R/O
This is the Maximum Phase Deviation.	<0,100>		VT_R8	N/A
Example:	8.0			
This is a parameter setting in **Single Target Detection** dialog box. Several single targets occurring at the same range will give you echoes in different parts of the beam's cross section. To remove the bad targets, the angle (phase) between the samples in the echo are measured. If the angle is too large, the echoes are deleted. This setting applies to both CW and FM operation.				

MinSpacing

Description	Range	Unit	Type	R/O
This is the minimum acceptable spacing between two targets.	<0,5>		VT_R8	N/A
Example:	0.0			
Supported in REST API only. This is a parameter setting in **Single Target Detection** dialog box. If you set **Minimum Echo Spacing** to 0.5, all echoes closer than 0.5 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

Format-type

Description	Range	Unit	Type	R/O
Message format.	EK60 EK50 HAC		VT_BSTR	N/A
Example:	EK60			
Supported in REST API only.				

Output

```
struct StructTSDataHeader
  {
  DWORDLONG dlTime;
  };
struct StructEchoTrace
  {
  double Depth; // (m) This is the depth of the target.
  double TSComp; // (dB) This is the compensated
Target Strength (TS) value.
  double TSUncomp; // (dB) This is the uncompensated
Target Strength (TS) value.
  double AlongshipAngle; // (deg) This is the
angle offset in the alongship direction.
  double AthwartshipAngle; // (deg) This is the
angle offset in the athwartship direction.
  double sa; // In this context the sA value presents
the current biomass index.
  };
struct StructTSDataBody
  {
  WORD NoOfEchoTraces; // This information identifies
the number of individual targets (single fish) in the depth layer.
  StructEchoTrace EchoTraceElement[100];
  };
struct StructTSData
  {
  StructTSDataHeader TSDataHeader;
  StructTSDataBody TSDataBody;
  };
```

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

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

```
MaxGainCompensation=6.0,
MaxPhasedeviation=7.0
```

Related topics

Data subscription types, page 40

Data subscription type: Target Strength (TS) detection chirp

The target strength (TS) detection chirp subscription provides single targets detected from an FM pulse. The target strength (TS) detection chirp data subscription string is **TSDetectionChirp**.

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

LayerType

Description	Range	Unit	Type	R/O
This is the current layer type.	Surface Bottom Pelagic		VT_BSTR	N/A
Example:	Surface			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

MinTSValue

Description	Range	Unit	Type	R/O
This is the Minimum Target Strength (TS) Value, also known as Minimum Threshold.	<-120,50>	dB	VT_R8	N/A
Example:	–50.0			
This is a parameter setting in **Single Target Detection** dialog box. The target strength (TS) for a single target must exceed this threshold (in dB) to be accepted. This setting applies to both CW and FM operation.				

RangeBeforeTarget

Description	Range	Unit	Type	R/O
This is the **Distance Before Target** setting.	<0,5>	m	VT_R8	N/A
Example:	0.15			
This is a parameter setting in **Single Target Detection** dialog box. The **Distance Before Target** and **Distance After Target** settings define the required spacing before and after one target to the end and beginning of the next target. The value for the **Distance After Target** should normally be larger than the value for **Distance Before Target** due to the backscattering properties of a target. This setting only applies to FM operation.				

RangeAfterTarget

Description	Range	Unit	Type	R/O
This is the **Distance After Target** setting.	<0,5>	m	VT_R8	N/A
Example:	0.15			
This is a parameter setting in **Single Target Detection** dialog box. The **Distance Before Target** and **Distance After Target** settings define the required spacing before and after one target to the end and beginning of the next target. The value for the **Distance After Target** should normally be larger than the value for **Distance Before Target** due to the backscattering properties of a target. This setting only applies to FM operation.				

MaxGainCompensation

Description	Range	Unit	Type	R/O
This is the Maximum Gain Compensation.	<0,12>		VT_R8	N/A
Example:	6.0			
This is a parameter setting in **Single Target Detection** dialog box. Targets located off centre will offer weaker echoes due to the beam properties. The EK80 system automatically compensates for this using a mathematical model, and you can manually control the effect of this algorithm by defining a maximum gain value. This setting applies to both CW and FM operation.				

MaxPhaseDeviation

Description	Range	Unit	Type	R/O
This is the Maximum Phase Deviation.	<0,100>		VT_R8	N/A
Example:	8.0			
This is a parameter setting in **Single Target Detection** dialog box. Several single targets occurring at the same range will give you echoes in different parts of the beam's cross section. To remove the bad targets, the angle (phase) between the samples in the echo are measured. If the angle is too large, the echoes are deleted. This setting applies to both CW and FM operation.				

Format-type

Description	Range	Unit	Type	R/O
Message format.	EK60 EK50 HAC		VT_BSTR	N/A
Example:	EK60			
Supported in REST API only.				

Output

```
struct StructEchoTraceWBT
  {
  double Depth; // (m) This is the depth of the target.
  double TSComp; // (dB) This is the compensated
Target Strength (TS) value.
  double TSUncomp; // (dB) This is the uncompensated
Target Strength (TS) value.
  double AlongshipAngle; // (deg) This is the
angle offset in the alongship direction.
  double AthwartshipAngle; // (deg): This is the
angle offset in the athwartship direction.
  double sa; // In this context the sA value presents
the current biomass index.
  float frequencyLimits[2];
  float uncompensatedFrequencyResponse[1000];
  float compensatedFrequencyResponse[1000];
  int withinMaxBeamCompensation[1000];
  };
struct StructTSDataWBT
  {
  DWORDLONG dlTime;
  WORD NoOfEchoTraces;
  StructEchoTraceWBT EchoTraceElement[100];
  };
```

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
TSDetectionChirp,
ChannelID=<ChannelID>,
LayerType=Surface,
Range=200,
RangeStart=3,
MinTSValue=-55,
RangeBeforeTarget=0.15,
RangeAfterTarget=0.7,
MaxGainCompensation=6.0,
MaxPhasedeviation=7.0
```

Related topics
Data subscription types, page 40

Data subscription type: Targets echogram

The targets echogram data subscription string is **TargetsEchogram**.

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

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

PixelCount

Description	Range	Unit	Type	R/O
This is the number of pixels/values to be returned by this subscription.	<0,10000>		VT_I4	N/A
Example:	500			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

TVGType

Description	Range	Unit	Type	R/O
This is the Time Variable Gain (TVG) parameter setting.	Pr Sv Sp TS SpAndTS		VT_BSTR	N/A
Example:	Sv			

EchogramType

Description	Range	Unit	Type	R/O
This is the echogram type in use.	Surface Bottom Trawl		VT_BSTR	N/A
Example:	Surface			

MinTSValue

Description	Range	Unit	Type	R/O
This is the Minimum Target Strength (TS) Value, also known as Minimum Threshold.	<-120,50>	dB	VT_R8	N/A
Example:	–50.0			
This is a parameter setting in **Single Target Detection** dialog box. The target strength (TS) for a single target must exceed this threshold (in dB) to be accepted. This setting applies to both CW and FM operation.				

MinEchoLength

Description	Range	Unit	Type	R/O
This is the Minimum Echo Length,	<0,20>		VT_R8	N/A
Example:	0.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Minimum Echo Length** to 0.8, all echoes shorter than 0.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxEchoLength

Description	Range	Unit	Type	R/O
This is the Maximum Echo Length.	<0,20>		VT_R8	N/A
Example:	1.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Maximum Echo Length** to 1.8, all echoes longer than 1.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxGainCompensation

Description	Range	Unit	Type	R/O
This is the Maximum Gain Compensation.	<0,12>		VT_R8	N/A
Example:	6.0			
This is a parameter setting in **Single Target Detection** dialog box. Targets located off centre will offer weaker echoes due to the beam properties. The EK80 system automatically compensates for this using a mathematical model, and you can manually control the effect of this algorithm by defining a maximum gain value. This setting applies to both CW and FM operation.				

MaxPhaseDeviation

Description	Range	Unit	Type	R/O
This is the Maximum Phase Deviation.	<0,100>		VT_R8	N/A
Example:	8.0			
This is a parameter setting in **Single Target Detection** dialog box. Several single targets occurring at the same range will give you echoes in different parts of the beam's cross section. To remove the bad targets, the angle (phase) between the samples in the echo are measured. If the angle is too large, the echoes are deleted. This setting applies to both CW and FM operation.				

CompressionType

Description	Range	Unit	Type	R/O
This is the interpolation method. If the number of samples is larger than the requested number of pixels, select how to interpolate.	Mean Peak		VT_BSTR	N/A
Example:	Mean			

ExpansionType

Description	Range	Unit	Type	R/O
This is the extrapolation method. If the number of samples is less than the requested number of pixels, select how to extrapolate.	Interpolation Copy		VT_BSTR	N/A
Example:	Interpolation			

Output

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

- `nEchogramElement[30000]`: This is the echogram information on a 16-bit logarithmic format.

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

```
TargetsEchogram,
ChannelID=<ChannelID>,
PixelCount=500,
```

```
Range=200,
RangeStart=3,
TVGType=TS,
EchogramType=Surface,
MinTSValue=-55.0,
MinEcholength=0.7,
MaxEcholength=2.0,
MaxGainCompensation=6.0,
MaxPhasedeviation=7.0
```

Related topics

Data subscription types, page 40

Data subscription type: Targets integration

The targets integration data subscription string is **TargetsIntegration**.

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

Input

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- N/A = not applicable
- If no value is provided in the XML input string, the example value in each table is used as default.

LayerType

Description	Range	Unit	Type	R/O
This is the current layer type.	Surface Bottom Pelagic		VT_BSTR	N/A
Example:	Surface			

IntegrationState

Description	Range	Unit	Type	R/O
This is the "on/off" switch for the integration.	Start Stop		VT_BSTR	N/A
Example:	Start			

Update

Description	Range	Unit	Type	R/O
This is a "selector" that allows you to provide integration value for latest ping or for the calculation interval.	UpdatePing UpdateAccumulate		VT_BSTR	N/A
Example:	UpdatePing			

Range

Description	Range	Unit	Type	R/O
This is the range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	500			

RangeStart

Description	Range	Unit	Type	R/O
This is the start range for the subscription.	<0,10000>	m	VT_I4	N/A
Example:	10			
With **Bottom** layer type, **RangeStart** is relative to the detected bottom. In this case, negative values are allowed and used to set up a layer starting above the detected bottom.				

Margin

Description	Range	Unit	Type	R/O
This is a range selector. Use it to set the range over the detected bottom. This range will not be included in integration.	<0,200>	m	VT_I4	N/A
Example:	1			

SvThreshold

Description	Range	Unit	Type	R/O
This is a threshold selector. Set this threshold to remove the weakest echoes from the integration.	<-200,100>	dB	VT_R8	N/A
Example:	-100			

MinTSValue

Description	Range	Unit	Type	R/O
This is the Minimum Target Strength (TS) Value, also known as Minimum Threshold.	<-120,50>	dB	VT_R8	N/A
Example:	–50.0			
This is a parameter setting in **Single Target Detection** dialog box. The target strength (TS) for a single target must exceed this threshold (in dB) to be accepted. This setting applies to both CW and FM operation.				

MinEchoLength

Description	Range	Unit	Type	R/O
This is the Minimum Echo Length,	<0,20>		VT_R8	N/A
Example:	0.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Minimum Echo Length** to 0.8, all echoes shorter than 0.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxEchoLength

Description	Range	Unit	Type	R/O
This is the Maximum Echo Length.	<0,20>		VT_R8	N/A
Example:	1.8			
This is a parameter setting in **Single Target Detection** dialog box. If you set **Maximum Echo Length** to 1.8, all echoes longer than 1.8 times the length of the transmitted pulse will be deleted. This setting only applies to CW operation.				

MaxGainCompensation

Description	Range	Unit	Type	R/O
This is the Maximum Gain Compensation.	<0,12>		VT_R8	N/A
Example:	6.0			
This is a parameter setting in **Single Target Detection** dialog box. Targets located off centre will offer weaker echoes due to the beam properties. The EK80 system automatically compensates for this using a mathematical model, and you can manually control the effect of this algorithm by defining a maximum gain value. This setting applies to both CW and FM operation.				

MaxPhaseDeviation

Description	Range	Unit	Type	R/O
This is the Maximum Phase Deviation.	<0,100>		VT_R8	N/A
Example:	8.0			
This is a parameter setting in **Single Target Detection** dialog box. Several single targets occurring at the same range will give you echoes in different parts of the beam's cross section. To remove the bad targets, the angle (phase) between the samples in the echo are measured. If the angle is too large, the echoes are deleted. This setting applies to both CW and FM operation.				

Output

```
struct StructIntegrationDataHeader
  {
  DWORDLONG dlTime;
  };
struct StructIntegrationDataBody
  {
  double dSa;
  };
struct StructIntegrationData
  {
  StructIntegrationDataHeader IntegrationDataHeader;
  StructIntegrationDataBody IntegrationDataBody;
  };
```

- `dSa` (m^2/nmi^2): In this context the $\mathbf{s_A}$ value presents the current biomass index.

For description of output when subscribing data from the REST API, refer to *sounder_datagrams.h* and *sounder.proto*. After installing the EK80 23.6.0 software, you will find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Subscription string example

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

Related topics

Data subscription types, page 40

Parameter descriptions

Topics

About parameters

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

Parameters identified as "read only" (R/O) can not be set. Some of them, for example sensor parameters, can be set, but the new value will immediately be overwritten if an active sensor is connected.

The parameter name must be used when working with parameters as described in section *Parameter management*.

The following parameter types are used:

- Application parameters
- Operation control parameters
- Sample storage control parameters
- Navigation and vessel motion parameters
- Environment parameters (Water column)
- Environment parameters (Transducer face)
- Transceiver information parameters
- Ping based parameters (Conditions)
- Ping based parameters (Settings)
- Bottom detection parameters

Related topics

Parameter descriptions, page 84

Application parameters

The application parameters are used to identify the application, in this case the EK80.

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only

RemoteCommandDispatcher / BrowseInfoProvider / ApplicationName

Description	Range	Unit	Type	R/O
This is the name of the current application.	EK80		VT_BSTR	Yes
Example:	EK80			

RemoteCommandDispatcher / BrowseInfoProvider / ApplicationType

Description	Range	Unit	Type	R/O
This is the type of the current application.	EK80		VT_BSTR	Yes
Example:	EK80			

RemoteCommandDispatcher / BrowseInfoProvider / ApplicationDescription

Description	Range	Unit	Type	R/O
This is a short description of the current application.			VT_BSTR	Yes
Example:	EK80 Wide band scientific echo sounder			

RemoteCommandDispatcher / BrowseInfoProvider / ApplicationVersion

Description	Range	Unit	Type	R/O
This is the software version of the current application.	x.x.x.x		VT_BSTR	Yes
Example:	23.6.0			

Related topics

Parameter descriptions, page 84

Operation control parameters

The operation control parameters are those that control the basic operation of the EK80.

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - VT_I4 = Integer 4 bytes
- R/O = Read only

OperationControl / MainOperation

Description	Range	Unit	Type	R/O
This parameter specifies the main operating mode.	Inactive Normal Replay		VT_BSTR	No
Example:	Normal			

OperationControl / PlayState

Description	Range	Unit	Type	R/O
This parameter indicates if the EK80 is running (pinging) or if it has stopped.	Stopped Running		VT_BSTR	No
Example:	Running			

AcousticDeviceSynchroniser / PingMode

Description	Range	Unit	Type	R/O
Ping Mode	Single step Interval Maximum		VT_BSTR	No
Example:	Interval			

AcousticDeviceSynchroniser / Interval

Description	Range	Unit	Type	R/O
Ping Interval	<10,2E9>	ms	VT_I4	No
Example:	1000			

AcousticDeviceSynchroniser / ExtSyncMode

Description	Range	Unit	Type	R/O
This parameter specifies if and how the transmissions are synchronized.	StandAlone Master Slave		VT_BSTR	No
Example:	StandAlone			

In order to make the EK80 operate in *Slave* mode using the AUXILIARY connector on the Wide Band Transceiver (WBT), you need to set `AcousticDeviceSynchroniser / ExtSyncMode` to *StandAlone* and `TransceiverManager / ExternalTrig` to *1*.

AcousticDeviceSynchroniser / SyncDelay

Description	Range	Unit	Type	R/O
This parameter controls the synchronization delay.	<0,1000>	ms	VT_I4	No
Example:	StandAlone			

In *Master* mode, the EK80 system waits for the delay time after the external trigger signal has been sent to the slaves before transmitting the ping. This is often referred to as a *pre-trigger*. In *Slave* mode, the EK80 system waits for the delay time after the external trigger signal has arrived before transmitting the ping. This is often referred to as a *post-trigger*.

StateManager / ActivateState

Description	Range	Unit	Type	R/O
This parameter can be used to change the current user setting.			VT_BSTR	No
Example:	User 3			

Related topics

Parameter descriptions, page 84

Sample storage control parameters

These parameters are those that control the raw data recording in the EK80.

- Type = The variant type as defined for certain programming languages
  - VT_I4 = Integer 4 bytes
- R/O = Read only

SounderStorageManager / SaveRawData

Description	Range	Unit	Type	R/O
This parameter controls if recording of raw data is on or off.	1 = On 0 =		VT_I4	No
Example:	1			

SounderStorageManager / SampleRange

Description	Range	Unit	Type	R/O
This parameter controls the depth range to be included in the raw data recording.	<0,10000>	m	VT_I4	No
Example:	1000			

SounderStorageManager / SampleRangeAuto

Description	Range	Unit	Type	R/O
This parameter sets the depth range to *Auto*.	1 = On 0 =		VT_I4	No
Example:	1			

SounderStorageManager / IndividualChannelRecordingRange

Description	Range	Unit	Type	R/O
This parameter specifies if the depth range for the recording is common for all channels (On), or individual for each channel (Off).	1 = On 0 =		VT_I4	No
Example:	0			

SounderStorageManager / <ChannelID> / Range

Description	Range	Unit	Type	R/O
If the depth range is selected to be individual for each channel, this parameter sets the range for specified channel.	<0,10000>	m	VT_I4	No
Example:	500			

SounderStorageManager / <ChannelID> / RangeAuto

Description	Range	Unit	Type	R/O
If the depth range is selected to be individual for each channel, this parameter sets the range to *Auto* for specified channel.	1 = On 0 =		VT_I4	No
Example:	0			

Related topics

Parameter descriptions, page 84

Navigation and vessel motion parameters

These parameters describe the information from the relevant navigation and motion sensors connected to the EK80.

- Type = The variant type as defined for certain programming languages
  - V_R8 = double, 8 bytes
- R/O = Read only

Note

None of the parameters are "read only".

OwnShip / Speed

Description	Range	Unit	Type	R/O
This is the vessel's speed.	<0,100>	m/s	VT_R8	No
Example:	5.6			

OwnShip / Heading

Description	Range	Unit	Type	R/O
This is the vessel heading.	<-180,180>	deg	VT_R8	No
Example:	17.5			

OwnShip / Course

Description	Range	Unit	Type	R/O
This is the vessel's course.	<-180,180>	deg	VT_R8	No
Example:	17.4			

OwnShip / Latitude

Description	Range	Unit	Type	R/O
Vessel latitude	<-90,90>	deg	VT_R8	No
Example:	64.353			

OwnShip / Longitude

Description	Range	Unit	Type	R/O
This is the vessel's geographical longitude.	<-180,180>	deg	VT_R8	No
Example:	5.23			

OwnShip / Heave

Description	Range	Unit	Type	R/O
This is the vessel's heave.	<-100,100>	m	VT_R8	No
Example:	-0.3			

OwnShip / Roll

Description	Range	Unit	Type	R/O
This is the vessel's roll.	<-90,90>	deg	VT_R8	No
Example:	2.1			

OwnShip / Pitch

Description	Range	Unit	Type	R/O
This is the vessel's pitch.	<-90,90>	deg	VT_R8	No
Example:	1.3			

OwnShip / MotionData

Description	Range	Unit	Type	R/O
This is a combined array parameter that contains vessel roll, pitch, heave and heading.	<-90,90>	deg	VT_R8 VT_ARRAY	No
	<-90,90>	deg		
	<-100,100>	m		
	<-180,180>	deg		
Example:	2.1, 1.3, -0.3, 17.5			

OwnShip / VesselDistance

Description	Range	Unit	Type	R/O
This is the vessel's sailed distance.	<0,1E9>	nmi	VT_R8	No
Example:	1.3			

VesselDraft

Description	Range	Unit	Type	R/O
This is the vertical distance between the ship origin and the keel.		m	VT_R8	No
Example:				

WaterLevelDraft

Description	Range	Unit	Type	R/O
This is the vertical distance between the ship origin and the water level.		m	VT_R8	No
Example:				

DropKeelOffset

Description	Range	Unit	Type	R/O
The drop keel offset value is the vertical distance from the "upper" to the "lower" position of the drop keel.	<-100,100>	m	VT_R8	No
Example:	0			

In the EK80 user interface, the **Offset** parameters recorded during the transducer installation are used to place the transducer in the vessel's coordinate system, relative to the ship origo. The drop keel offset distance is automatically added to the transducer's vertical X-offset.

Related topics
Parameter descriptions, page 84

Environment parameters (Water column)

These parameter describe the sound propagation properties of the water column.

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only

OwnShip / EnvironmentData / WaterQuality

Description	Range	Unit	Type	R/O
This parameter specifies if the EK80 operates in fresh or salt water.	0 = Fresh Water 1 = Salt Water		VT_I4	No
Example:	1			

OwnShip / EnvironmentData / Temperature

Description	Range	Unit	Type	R/O
This parameter defines the water temperature. The value is used in the environment calculations.	<0,40>	deg C	VT_R8	No
Example:	7			

OwnShip / EnvironmentData / Salinity

Description	Range	Unit	Type	R/O
This parameter defines the water salinity. The value is used in the environment calculations.	<0,40>	PSU	VT_R8	No
Example:	8.0			

OwnShip / EnvironmentData / Acidity

Description	Range	Unit	Type	R/O
This parameter defines the water acidity. The value is used in the environment calculations.	<7.8,8.2>	pH	VT_R8	No
Example:	8.0			

OwnShip / EnvironmentData / DepthForSVandAbsorptionCalculations

Description	Range	Unit	Type	R/O
This parameter defines the water depth. The value is used in the Sv and absorption calculations..	<0,10000>	m	VT_R8	No
Example:	100			

OwnShip / EnvironmentData / LatitudeForPressureCalculations

Description	Range	Unit	Type	R/O
Vessel latitude The value is used in the the pressure calculations.	<0,90>	deg	VT_R8	No
Example:	45			

OwnShip / EnvironmentData / SoundVelocity

Description	Range	Unit	Type	R/O
This parameter defines the sound speed. The value is used in the environment calculations.	<1390,1700>	m/s	VT_R8	No
Example:	1483.9			

OwnShip / EnvironmentData / SoundVelocitySource

Description	Range	Unit	Type	R/O
This parameter specifies if the sound speed is inserted manually, or if it is calculated by the EK80 based on the other parameters.	Manual Calculated		VT_BSTR	No
Example:	Calculated			

Related topics

Parameter descriptions, page 84

Environment parameters (Transducer face)

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only

OwnShip / EnvironmentData / SoundVelocityTransducerSource

Description	Range	Unit	Type	R/O
This parameter specifies if the sound speed is inserted manually, or if it is imported from a sound speed sensor.	Profile Manual		VT_BSTR	No
Example:	Profile			

OwnShip / EnvironmentData / SoundVelocityTransducer

Description	Range	Unit	Type	R/O
This parameter defines the sound speed. The value is used in the environment calculations.	<1390,1700>	m/s	VT_R8	No
Example:	1483.9			

Related topics

Parameter descriptions, page 84

Transceiver information parameters

These parameters identifies the individual transceivers that are used in the EK80 system.

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
- R/O = Read only

TransceiverMgr / Transceivers

Description	Range	Unit	Type	R/O
This parameter provides a list of the transceivers used by the EK80 system. The list includes transceiver type (abbreviated) and serial number.			VT_BSTR VT_ARRAY	Yes
Example:	WBT 5197648			

TransceiverMgr / <TransceiverID> / TransceiverType

Description	Range	Unit	Type	R/O
For the given <TransceiverID>, this parameter identifies the type of transceiver in use.			VT_BSTR	Yes
Example:	WBT			

- "GPT" identifies the "General Purpose Transceiver"
- "WBT" identifies the "Wide Band Transceiver"
- "SBT" identifies the "Single Band Transceiver"
- "WBT Mini" identifies the miniature wide band echo sounder transceiver
- "WBT Tube" identifies the subsea wide band echo sounder transceiver
- "EC150" identifies the ADCP and echo sounder transceiver

TransceiverMgr / <TransceiverID> / SerialNumber

Description	Range	Unit	Type	R/O
For the given <TransceiverID>, this parameter identifies the serial number of the transceiver in use.			VT_I4	Yes
Example:	5197648			

Related topics
Parameter descriptions, page 84

Ping based parameters (Conditions)

These ping based parameters identify the information from the relevant navigation and motion sensors connected to the EK80 at the time of the latest transmission ("ping").

- Type = The variant type as defined for certain programming languages
  - V_R8 = double, 8 bytes
- R/O = Read only

TransceiverMgr / PingTime

Description	Range	Unit	Type	R/O
This parameter specifies the time of the latest ping. The time is given in nanoseconds since 1601.			VT_CY	Yes
Example:	127510454149223493			

TransceiverMgr / Latitude

Description	Range	Unit	Type	R/O
Vessel latitude	<-90,90>	deg	VT_R8	Yes
Example:	64.353			

TransceiverMgr / Longitude

Description	Range	Unit	Type	R/O
This is the vessel's geographical longitude.	<-180,180>	deg	VT_R8	Yes
Example:	5.23			

TransceiverMgr / Heave

Description	Range	Unit	Type	R/O
This is the vessel's heave.	<-100,100>	m	VT_R8	Yes
Example:	-0.3			

TransceiverMgr / Roll

Description	Range	Unit	Type	R/O
This is the vessel's roll.	<-90,90>	deg	VT_R8	Yes
Example:	2.1			

TransceiverMgr / Pitch

Description	Range	Unit	Type	R/O
This is the vessel's pitch.	<-90,90>	deg	VT_R8	Yes
Example:	1.3			

TransceiverMgr / VesselDistance

Description	Range	Unit	Type	R/O
This is the vessel's sailed distance.	<0,1E9>	nmi	VT_R8	Yes
Example:	1.3			

MRUHeading

Description	Range	Unit	Type	R/O
This is the vessel heading provided by the motion reference unit (MRU). It is often referred to as "yaw".	<0,360>		VT_R8	Yes
Example:	10.7			

ProcessingMgr / <ChannelID> / NoiseEstimate

Description	Range	Unit	Type	R/O
This is the noise estimate for the specified channel. In this context, the term *channel* is used as a common term to identify the combination of transceiver, transducer and operating frequency.	<-200,0>	dB	VT_R8	Yes
Example:	-123.4			

Related topics

Parameter descriptions, page 84

Ping based parameters (Settings)

These ping based parameters identify the transceiver settings at the time of the latest transmission ("ping").

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only
- Coordinate system
  - Alpha = Athwartship
  - Beta = Alongship

TransceiverMgr / Channels

Description	Range	Unit	Type	R/O
This parameter specifies which channels that are used on the transceivers. The list includes transceiver type, serial number and transducer name for each channel.			VT_BSTR VT_ARRAY	Yes
Example:	WBT 5197648-15 ES120-7C			

TransceiverMgr / <ChannelID> / ChannelMode

Description	Range	Unit	Type	R/O
This is the current transceiver mode.	0 = Active 1 =		VT_I4	No
Example:	0			

TransceiverMgr / <ChannelID> / TransmitPower

Description	Range	Unit	Type	R/O
This is the transmit power. The power maximum rating depends on the transceiver type.	<0,10000>	W	VT_R8	No
Example:	250			

TransceiverMgr / <ChannelID> / PulseForm

Description	Range	Unit	Type	R/O
This is the pulse type.	0 = CW 1 = FM 5 = FMD		VT_I4	No
Example:	1			

TransceiverMgr / <ChannelID> / Frequency

Description	Range	Unit	Type	R/O
This is nominal frequency for CW transmissions.	<1000,1E6>	Hz	VT_R8	No
Example:	120000			

TransceiverMgr / <ChannelID> / FrequencyStart

Description	Range	Unit	Type	R/O
This is the start frequency in sweep. The frequency range is transducer dependent.		Hz	VT_R8	No
Example:	90000			

TransceiverMgr / <ChannelID> / FrequencyEnd

Description	Range	Unit	Type	R/O
This is the end frequency in sweep. The frequency range is transducer dependent.		Hz	VT_R8	No
Example:	120000			

TransceiverMgr / <ChannelID> / PulseLength

Description	Range	Unit	Type	R/O
This is the pulse duration.	<0, 0.065535>	s	VT_R8	No
Example:	0.001024			

TransceiverMgr / <ChannelID> / RampingCW

Description	Range	Unit	Type	R/O
This is the pulse ramping for CW transmissions.	0 = Slow 1 = Fast		VT_I4	No
Example:	1			

TransceiverMgr / <ChannelID> / RampingFM

Description	Range	Unit	Type	R/O
This is the pulse ramping for FM transmissions.	0 = Slow 1 = Fast		VT_I4	No
Example:	1			

TransceiverMgr / <ChannelID> / SampleInterval

Description	Range	Unit	Type	R/O
This is the sample interval. The *sample rate* is the average number of samples obtained in one second. The *sample interval* is 1/*sample rate* to allow readout in time (normally milliseconds).	<0.000010 , 0.065535>	s	VT_R8	Yes
Example:	8e-06			

TransceiverMgr / <ChannelID> / AbsorptionCofficient

Description	Range	Unit	Type	R/O
This is the absorption coefficient. The value is based on the CW frequency or the centre of an FM frequency sweep.	<0, 0.3>	dB/m	VT_R8	Yes
Example:	0.05			

TransceiverMgr / <ChannelID> / SoundVelocity

Description	Range	Unit	Type	R/O
This parameter defines the sound speed. The value is used in the environment calculations.	<1390,1700>	m/s	VT_R8	Yes
Example:	1483.9			

TransceiverMgr / <ChannelID> / TransducerName

Description	Range	Unit	Type	R/O
This is the custom name of the transducer. The name is provided during the transducer installation in the EK80 user interface.			VT_BSTR	Yes
Example:	ES120-7C			

TransceiverMgr / <ChannelID> / SerialNumber

Description	Range	Unit	Type	R/O
This is the serial number of the transducer in use on the current channel. The information assumes that you have typed in the serial number when you installed the transducer in the user interface.			VT_I4	Yes
Example:	191			

TransceiverMgr / <ChannelID> / EquivalentBeamAngle

Description	Range	Unit	Type	R/O
This is the equivalent beam angle. It is commonly referred to as *Equivalent 2-way beam angle*.	<-100,0>	dB	VT_R8	Yes
Example:	-20.7			

TransceiverMgr / <ChannelID> / AngleSensitivityAlongship

Description	Range	Unit	Type	R/O
This is the angle sensitivity in the alongship direction.	<0,100>	El.deg/ Mech.deg	VT_R8	Yes
Example:	23			

TransceiverMgr / <ChannelID> / AngleSensitivityAthwartship

Description	Range	Unit	Type	R/O
This is the angle sensitivity in the athwartship direction.	<0,100>	El.deg/ Mech.deg	VT_R8	Yes
Example:	23			

TransceiverMgr / <ChannelID> / BeamWidthAlongship

Description	Range	Unit	Type	R/O
This is the beamwidth in the alongship direction.	<0,100>	deg	VT_R8	Yes
Example:	7			

TransceiverMgr / <ChannelID> / BeamWidthAthwartship

Description	Range	Unit	Type	R/O
This is the beamwidth in the athwartship direction.	<0,100>	deg	VT_R8	Yes
Example:	7			

TransceiverMgr / <ChannelID> / AngleOffsetAlongship

Description	Range	Unit	Type	R/O
This is the angle offset in the alongship direction.	<-10,10>	deg	VT_R8	Yes
Example:	0			

TransceiverMgr / <ChannelID> / AngleOffsetAthwartship

Description	Range	Unit	Type	R/O
This is the angle offset in the athwartship direction.	<-10,10>	deg	VT_R8	Yes
Example:	0			

TransceiverMgr / <ChannelID> / Gain

Description	Range	Unit	Type	R/O
This is the minimum level applied for the relevant channel.	<1,100>	dB	VT_R8	Yes
Example:	27			

TransceiverMgr / <ChannelID> / SaCorrection

Description	Range	Unit	Type	R/O
This is the sA correction.	<-10,10>	dB	VT_R8	Yes
Example:	0			

TransceiverMgr / ExternalTrig

Description	Range	Unit	Type	R/O
This parameter controls if the EK80 can operate in *Slave* mode using an external trigger.	1 = On 0 =		VT_I4	No
Example:	1			

In order to make the EK80 operate in *Slave* mode using the AUXILIARY connector on the Wide Band Transceiver (WBT), you need to set `AcousticDeviceSynchroniser / ExtSyncMode` to *StandAlone* and `TransceiverManager / ExternalTrig` to *1*.

Related topics

Parameter descriptions, page 84

Bottom detection parameters

These parameters identify the bottom detector settings at the time of the latest transmission ("ping").

- Type = The variant type as defined for certain programming languages
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
  - VT_I4 = Integer 4 bytes
- R/O = Read only

ProcessingMgr / Channels

Description	Range	Unit	Type	R/O
This parameter specifies which channels that are used on the transceivers. The list includes transceiver type, serial number and transducer name for each channel.			VT_BSTR VT_ARRAY	Yes
Example:	WBT 5197648-15 ES120-7C			

ProcessingMgr / <ChannelID> / ChannelProcessingCommon / BottomDetectionActive

Description	Range	Unit	Type	R/O
This parameter specifies if the bottom detection is off or on.	0 = Off 1 =		VT_I4	No
Example:	1			

Since each channel can be presented in multiple views the values for Upper Detection Limit, Lower Detection Limit and Bottom Backstep must be defined individually.

- `ProcessingMgr / <ChannelID> / ChannelProcessingCommon / UpperDetectorLimit`
- `ProcessingMgr / <ChannelID> / ChannelProcessingCommon / LowerDetectorLimit`
- `ProcessingMgr / <ChannelID> / ChannelProcessingCommon / BottomBackstep`

ProcessingMgr / <ChannelID> / ChannelProcessingCommon / UpperDetectorLimit

Description	Range	Unit	Type	R/O
This is the **Minimum Depth** setting used for bottom detection.	<0,10000>	m	VT_R8	No
Example:	10			

ProcessingMgr / <ChannelID> / ChannelProcessingCommon / LowerDetectorLimit

Description	Range	Unit	Type	R/O
This is the **Maximum Depth** setting used for bottom detection.	<0,10000>	m	VT_R8	No
Example:	1000			

ProcessingMgr / <ChannelID> / ChannelProcessingCommon / BottomBackstep

Description	Range	Unit	Type	R/O
This is the **Bottom Backstep** setting for the channel.	<-100,10>	dB	VT_R8	No
Example:	-50			

Related topics

Parameter descriptions, page 84

Ping mode manager parameters

These parameters identify the ADCP current velocity settings at the time of the latest transmission ("ping").

- Type = The variant type as defined for certain programming languages
  - VT_I4 = Integer 4 bytes
  - VT_BSTR = Text string
  - V_R8 = double, 8 bytes
- R/O = Read only

Note

*The **VirtualChannelID** comprises a **ChannelID** and a **PingID**.*

PingModeManager / EsOrAdcpPingMode

Description	Range	Unit	Type	R/O
This parameter forces all EC transceivers to either *ES* (echo sounder) mode or *ADCP* (current velocity measurement) mode	*ES* *ADCP* *Undefined*		VT_BSTR	No
Example:	*Undefined*			

PingModeManager / <ChannelID> / ADCP / PulseForm

Description	Range	Unit	Type	R/O
This parameter controls the pulse form for the ping transmission.	*CW* = Continuous Wave *LFM* = Low–Frequency Modulation *HFM* = High-Frequency Modulation User-defined *Ricker* *LFMD* =		VT_BSTR	No
Example:	*LFM*			

PingModeManager / <ChannelID> / ADCP / PulseBandWidth

Description	Range	Unit	Type	R/O
This parameter controls the sub-pulse bandwidth.	<0,24000>	Hz	VT_R8	No
Example:	0 (default)			

Note

This parameter is set to 0 Hz if PulseForm is set to CW (Continuous Wave).

PingModeManager / <ChannelID> / ADCP / DepthCellSize

Description	Range	Unit	Type	R/O
This parameter controls the size of depth cells for ADCP (2 |4 |8|16 m).	(2 |4 |8|16 m)	m	VT_R8	No
Example:	4 (default)			

PingModeManager / <ChannelID> / ADCP / MaxVesselSpeed

Description	Range	Unit	Type	R/O
This parameter controls the estimated maximum speed of the vessel during ADCP operations.	<0.0,10.0>	m/s	VT_R8	No
Example:	5 (default)			

Note

This parameter is not in use when EK80 is connected to an MRU using KM binary datagram.

PingModeManager / <ChannelID> / ADCP / MaxCurrentSpeed

Description	Range	Unit	Type	R/O
This parameter controls the estimated maximum current speed during ADCP operations.	<0.0,10.0>	m/s	VT_R8	No
Example:	5 (default)			

PingModeManager / <ChannelID> / ADCP / FrequencyTx

Description	Range	Unit	Type	R/O
This parameter controls an array of all ping frequencies.		Hz	VT_R8|VT_ ARRAY	No
Example:				

PingModeManager / <ChannelID> / ADCP / TxPower

Description	Range	Unit	Type	R/O
Tx Power	<0,500>	W	VT_R8	No
Example:	0 (default)			

Related topics

Parameter descriptions, page 84

REST API

Topics

What is a REST API?

API is short for Application Programming Interface and comprises a set of rules letting programs communicate with each other. REST stands for REpresentational State Transfer.

A REST API allows a client to request information from a server using HTTP commands in a similar way to requesting data from a website. In this context, the server is the EK80 software.The communication is standardized and includes features such as scalability and statelessness. As the complexity of the service grows, you can easily make modifications and do not have to keep track of the state of the data across client and server.

The REST APIs communicate using specific ports. There are two REST APIs described for the EK80.

- EK80 REST API for remote control

  This API is used for setting and receiving simple data and intended for test purposes only. This is a beta-version.

- EK80 REST API for data subscription.

  This is an alpha-version.

By default, they are both configured for a client application within the echo sounder computer. If they will be used from a computer inside the LAN, use the **WEB API Configuration** dialog box to configure the IP addresses.

Note

The REST API is currently under development and only limited support is provided.

Request and response using HTTP

The request and response is delivered using standard HTTP. The requests available are:

- Get
- Post
- Put
- Patch
- Delete

For EK80 systems the form-ended requests returns JSON-encoded (Javascript Object Nation) responses. The HTTP standard is open for other formats as well.

Based on these requests, your client can receive and manipulate data in the EK80 system.

Endpoints

An API endpoint is the point of entry in a communication channel when two systems are interacting. It refers to touchpoints of the communication between an API and a server. The location where the API sends a request and where the response emanates is called an endpoint.

Error codes

The API uses conventional HTTP response codes to indicate success or failure of an API request.

- **2xx range** indicate success
- **4xx range** indicate an error that failed given the information provided (a required parameter was omitted)
  - **400 - Bad Request**: The request was unacceptable, often due to missing required parameter.
  - **401 - Unauthorized**: No valid API key provided.
  - **402 -Request Failed**: The parameters were valid but the request failed.
  - **404 - Not Found**: The requested resource doesn't exist.
  - **409 - Conflict**: The request conflicts with another request (perhaps due to using the same idempotent key).
  - **429 - Too Many requests**: Too many requests hit the API too quickly. We recommend an exponential backoff of your requests.
- **5xx range** indicates error with the servers.
  - **500, 502,503, 504 - Server Errors**: Something went wrong on the server side.

What is Swagger?

The REST API for the EK80 echo sounder is documented using the Swagger tool.Swagger follows the OpenAPI Specification.

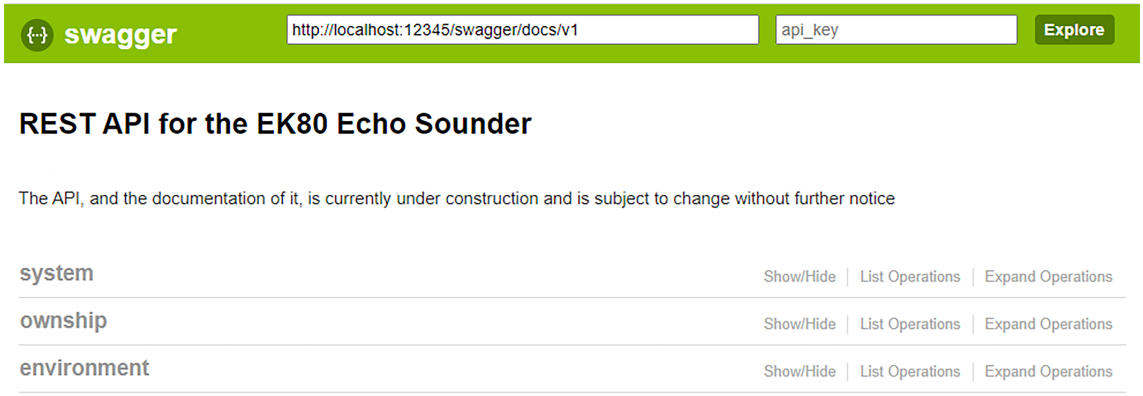
Your client application can communicate with the EK80 system using dedicated ports. The client interfaces defined at this stage and their defined port of communication is summarized in this list.

- http://localhost:12345/swagger EK80 Remote Control API
- http://localhost:12346/swagger EK80 Data subscription API

Note

You must start the EK80 software prior to opening Swagger for communication through the REST API. If the client accessing the API is located on a computer separate from the computer running the EK80 software, use the **WEB API Configuration** *dialog box to configure the IP addresses of the EK80 adapter where the client computer is connected.*

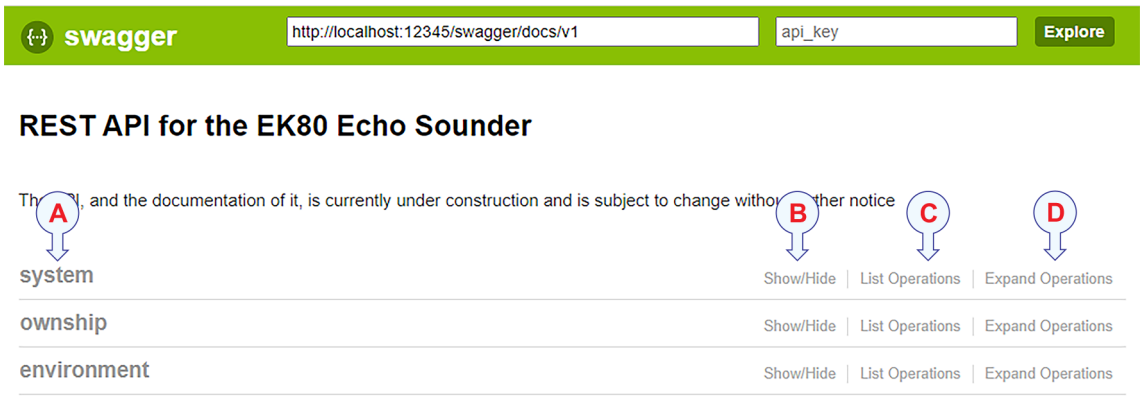
Entering this address in your favourite web browser will open the start page, and you can start sending/receiving requests/responses.



Working with REST API and Swagger

Browsing the API

The Swagger user interface at first glance provides an overview of the groups of operations which are defined.



A **Groups of operations**

Select any group of operations by clicking it the name and expand to see all operations in the group listed.

B **Show/Hide**

Select any operation group and expand/collapse the list of operations in the group by clicking **Show/Hide**

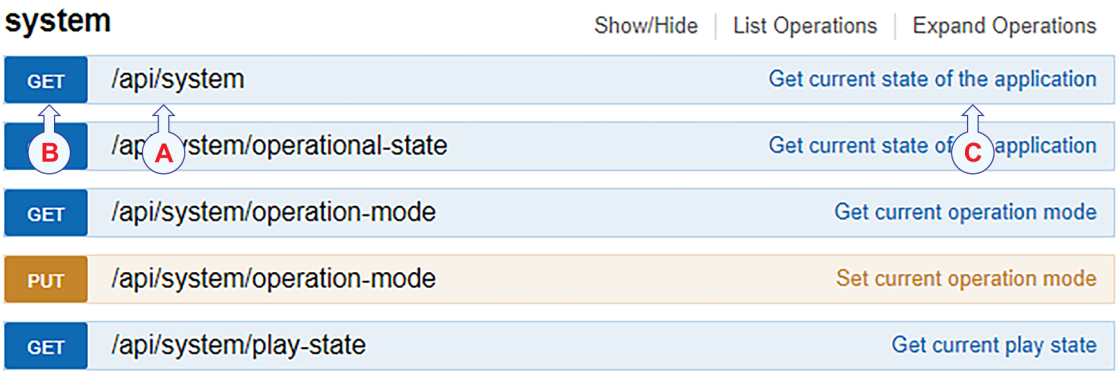
C **List operations**

Select List operations to expand any operation group to a full list of operations.

D **Expand Operations**

Selecting this option will expand the selected operation and display in detail request and response information.

Operation description



An operation in the Swagger user interface is defined by the following elements

A **Resource**

This is the descriptive name for the endpoint. "API" signifies that this is the API portion of the endpoint.

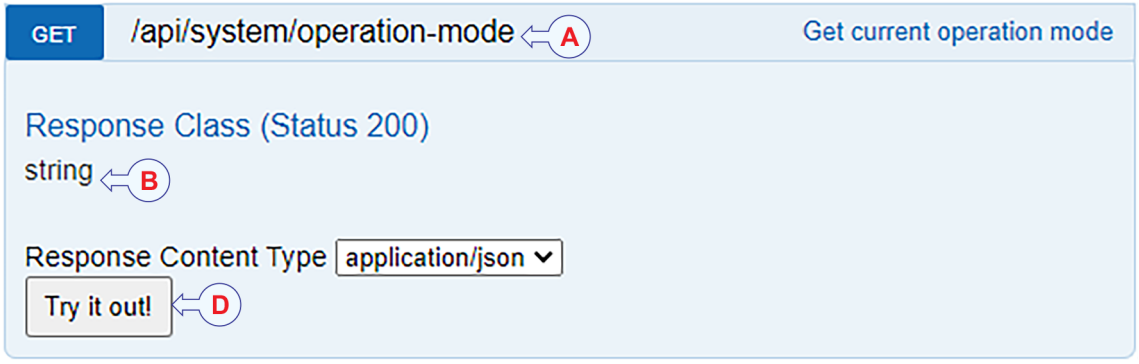
B **Method**

Defines the HTTP method that will be used to call the endpoint. (GET, POST, PUT, PATCH, DELETE).

C **Description**

Short description of the endpoint function.

Selecting one specific operation/parameter from the list, opens a more detailed view of the selected operation/parameter.



A URL of the parameter - indicates endpoint and resources.

B Parameter description

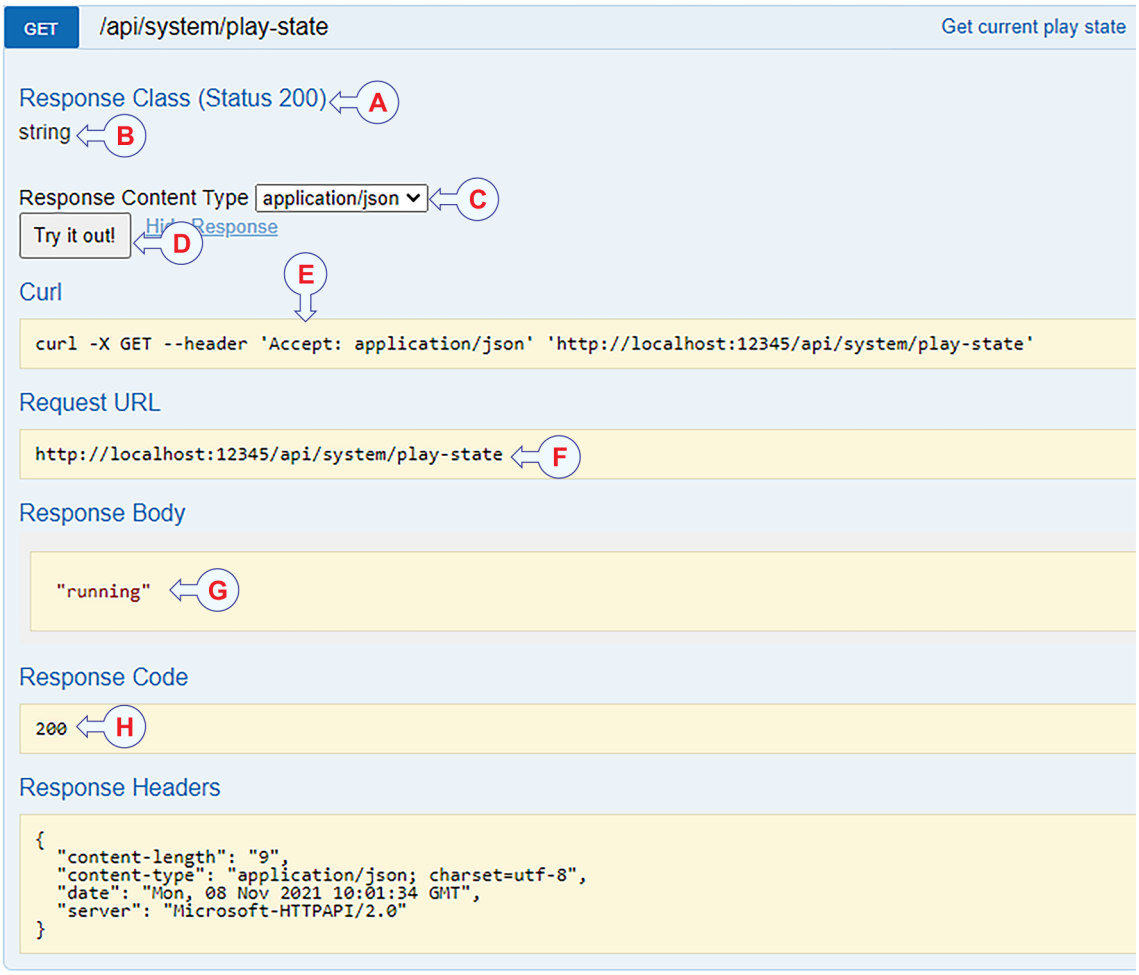
C Data type of the parameter

D **Try it out!** select this option to execute the request to the EK80 system.

Get a parameter

REST APIs communicate via HTTP requests to perform standard database functions such as creating, reading, updating and deleting records. within a resource. By selecting a specific operation in one of the operation groups you will be able to send a request and receive a response from the EK80 system.

Select **Try it out!** for a GET operation. The different elements for the request and response sequence is summed up in the Swagger user interface.



A **Response Class**

This element provides you with a quick information regarding the response using the HTTP standard error codes.

B **Response Content Type**

This element provides the formatting of the response body.

C **Response Messages**

This section displays the type of response which will be returned for the request. The response corresponds to standard HTTP status code.

D **Try it out!**

Select this to send the actual request.

E **Curl**

This element shows the cURL which is formats an URL into a string variable.

F **Request URL**

This element shows the actual URL.

G **Response Body**

The actual response "payload"

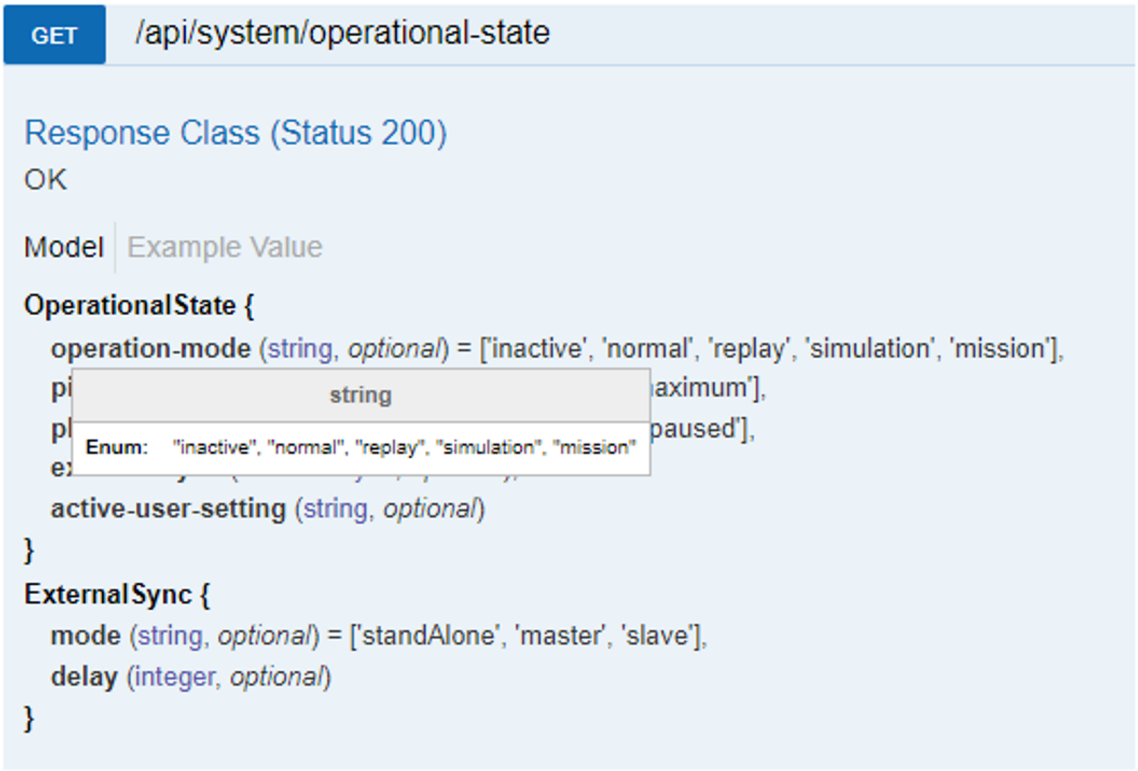
H **Response Code**

Standard response code from HTTP is used to indicate the status of a response.

Get a structure

Some of the parameters of the API is a composite structure returned as a json structure. For instance, will the `api/sounder/operation` URL return a structure which is a collection of the current state of all the "single-value" parameters of the "operation" node.

The structure and all the single value parameters is displayed underneath **Response Class**. Toggeling between **Model** and **Example Value** view will provide examples of parameter value as well as the full range of allowed parameter values. Selecting **Model** will also display the type of each parameter value.



Coordinate system

- Alpha = Athwartship
- Beta = Alongship

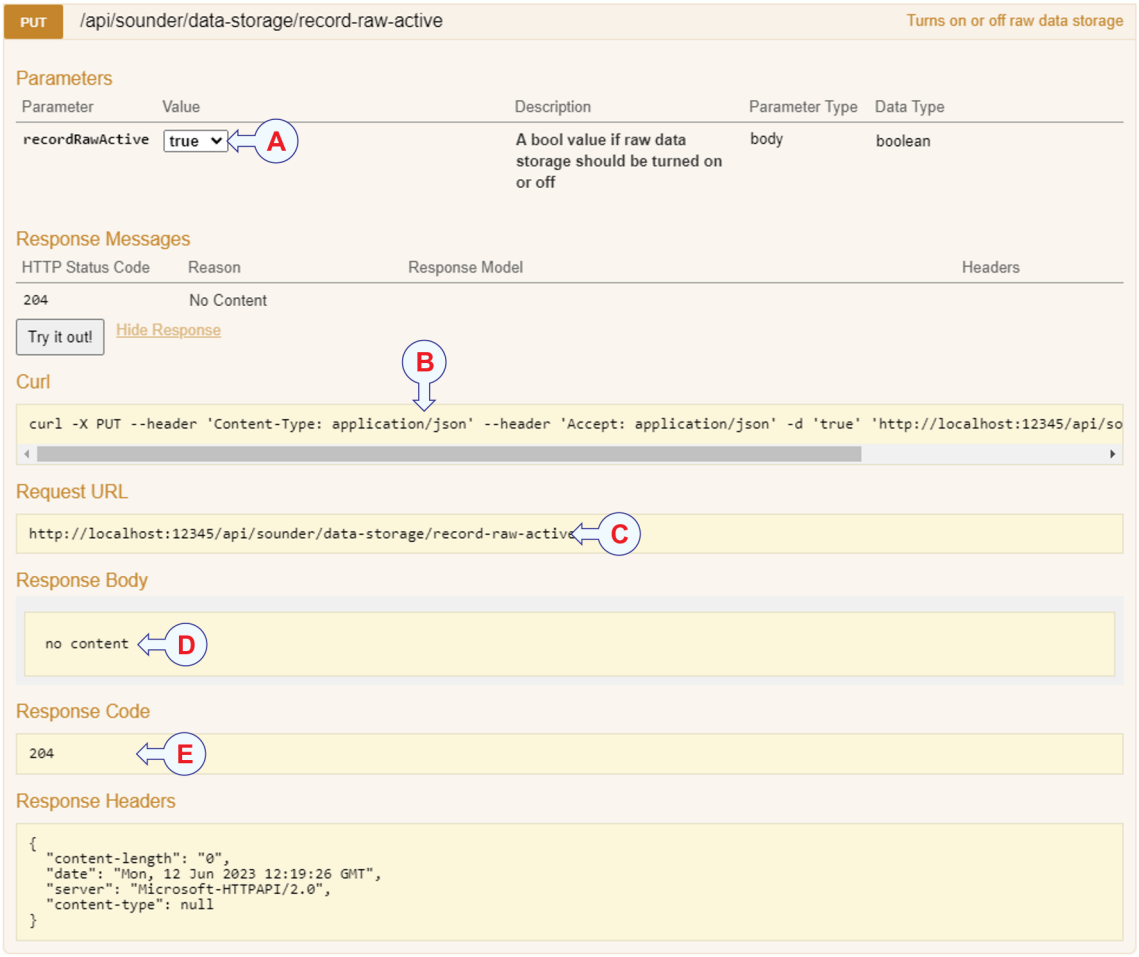
Set a parameter

Select a PUT operation and enter values for all parameters then press **Try it out!** button. The web browsers will then send a "PUT" request to the EK80 web server. The EK80 handles the request and makes a response back to the web browser.

Note

The PUT operation will change parameter settings in the EK80 software. Be sure to use legal parameter settings when performing this operation. Valid settings are described for each parameter in the Help-function in EK80 user interface

In the example below the **Raw Data Recording** functionality is on.



A Enter new parameter value in this field.

B cURL command line.

C The actual URL.

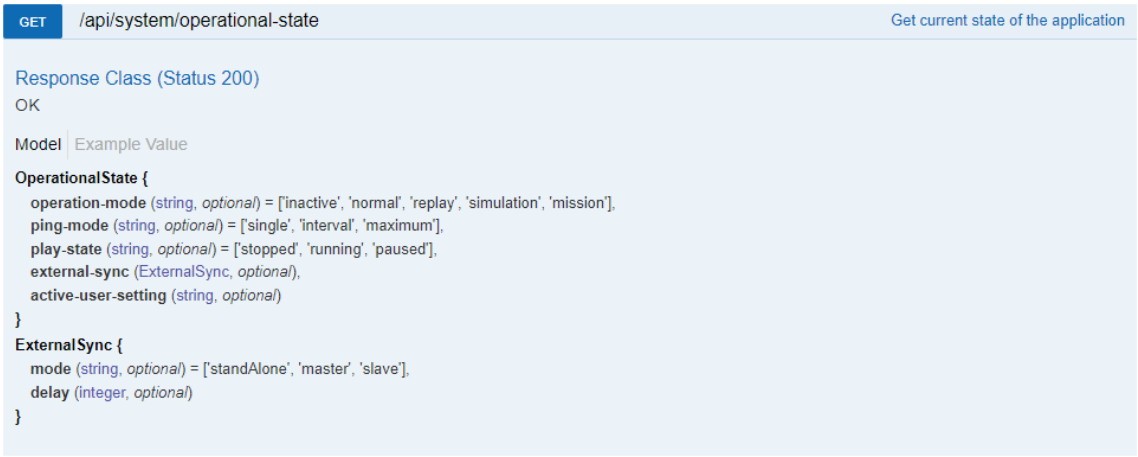
D The actual response payload.

E The HTTP response code. Response code in the 2xx range indicates a success in setting the parameter.

Legal parameter values

Enum-type parameters are specified as strings, and the composite structure parameters contain lists of legal values for the enum-type parameters used in the structure. The following URLs returns structures with one or more enum-types:

- /api/sounder/operation - returns current operation state & legal parameter values
- /api/sounder/environment - returns current environment state & legal parameter values
- /api/sounder/ping-configuration - returns current ping configuration states & legal values



The list of "legal" values can be found by pressing "Model" on a "structure" parameter.

Note

The Swagger documentation for this version of the REST API is under development and may not be complete. Please use the API the to find the exact parameters available.

Auto generation of client code

There are several tools available for automatically generate client code for a REST API. These tools use outputs of a running Swagger endpoint to automatically generate REST API code. Some examples of tools are **AutoREST** and **Visual Studio 2019**. The tools either connect to a running Swagger endpoint or read a *json* file with the "openAPI Specification" of the end point. The entire "OpenAPI Specification" for the EK80 endpoint can be accessed using the following URL:

http://localhost:12345/swager/docs/vl

The output from this URL can then be saved to a *json* file.

Adding a data subscription

The REST API can be used to create a stream event subscription. The EK80 system provides the possibility of subscribing to a variety of data. Once a subscription has been established, data will be sent to the subscription endpoint and clients can access it.

Context

Using the REST API requests and responses you can set up a subscription for EK80 data. List of supported and avaliable subscription types is found in the Swagger documentation.

In order to set up a subscription you will need to:

- Create a subscription.
- Create a communication endpoint.
- Add the subscription to the endpoint.

By splitting this process into three parts, you are able to add multiple subscriptions to one endpoint and to send one subscription to multiple endpoints.

Procedure

1 Start EK80 in *Normal* or *Replay* operating mode.

2 Acquire the **Channel ID** for the appropriate channel.

Note

The **Channel ID** *name may differ when using Normal mode versus using Replay mode, depending on the raw file.*

a Open the EK80 Subscription API.

The REST API is opened using Swagger documentation.

http://localhost:12345/swagger

b Find the **Channel ID**.

Under **ping-configuration** operations, perform a GET request:

`/api/sounder/ping-configuration/channel-list`

The **Response Body** will provide a full list of available channels in the form of their **Channel ID**. Example of a **Channel ID**:

```
"WBT 1034758-15 ES200-7C_ES"
```

c Make a note of the **Channel ID** for the channel you would like to subscribe to. this will be used to create a data subscription

3 Create the data subscription.

There are several output subscriptions available. As an example bottom detection is used in this procedure.

```
/api/sounder/data-output/bottom-detection-subscriptions
{
  "channel-id": "WBT 582156-15 ES120-7C_ES",
  "settings": {
    "upper-detector-limit": 10,
    "lower-detector-limit": 1000,
    "bottom-back-step": -50
  },
  "subscription-name": "BottomDepth",
  "subscriber-name": "ClientPC"
}
```

a Open the EK80 Subscription API.

The REST API is opened using Swagger documentation.

http://localhost:12346/swagger

b Locate the operation:

Create a bottom detection data subscription

`/api/sounder/data-output/bottom-detection-subscriptions`

c Fill in the specifications for the request.

Open the **Example Value** and right-click in the structure. This copies the structure into the specifications field for the request.

Also make sure the **Channel ID**, being a part of this request is correct.

d Enter appropriate `subscription-name` and `subscriber-name` values for the subscription in the subscription.
In this example **BottomDepth** and **ClientPC** has been selected.

e Select **Try it out!**
This creates the new subscription.

f Find the `subscription-id`.
In the **Response Body** of the response, you will find a number. This will be referred to as the `subscription-id`. Make a note of it. This will be used when adding a subscription to an endpoint.

4 Create a new communication endpoint.

a Locate the operation:

Create a new communication end point

`/api/sounder/data-output/communication-end-points`

b Fill in the specifications for the request.

```
{
  "name": "DepthOutput",
  "configuration": "tcp://10.120.65.47:20034",
  "communication-type": "zero-mq"
}
```

Open the **Example Value** and right-click in the structure. This copies the structure into the specifications field for the request.

Fill in a suitable name for the endpoint, use the IP address of one of the LAN adapters in the EK80 computer. Select a suitable/unique port.

c Select **Try it out!**.

d Find the `communication-end-point`.

In the **Response Body** of the response, you will find a number. This request is referred to as the `communication-end-point`. Make a note of it to use it when adding a subscription to the endpoint.

5 Send a request to add the subscription to the endpoint.

Add a subscription to the communication end point

`/api/sounder/data-output/communication-end-points/{endPointId}/data-subscriptions`

Use the `communication-end-point` to fill in **endPointId**.

```
{
  "subscription-id": 1,
  "data-serialization": "protobuf"
}
```

Open the **Example Value** and right-click in the structure. This copies the structure into the specifications field for the request.

Fill in the `subscription-id`.

The plan is to provide the subscribed data as C structure ("c-struct") and/or according to the Protocol Buffer standard ("protobuf"). Not all subscriptions may be supported on both formats. Refer to *sounder_datagrams.h* and *sounder.proto* files for the description of the subscriptions that are supported. You can find these definition files in `C:\ProgramData\Simrad\EK80\REST API`.

Note

Protocol Buffer (protobuf) is a Google mechanism for serializing structured data.

6 Select **Try it out!**.

The subscription is now connected to the endpoint and the port for the endpoint will provide data output for clients to use.

Note

The architecture of separated subscriptions and endpoints allows you to send multiple subscriptions to a single endpoint or to send a subscription to multiple endpoints.

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 Maritime. 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 Maritime 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 Maritime to contain configuration and depth information.

Related topics

Raw data format, page 119
Index file format for RAW data files, page 160
XYZ file format, page 162
The BOT data file format, page 163
NetCDF file format, page 162
The SEGY data file format, page 165

Raw data format

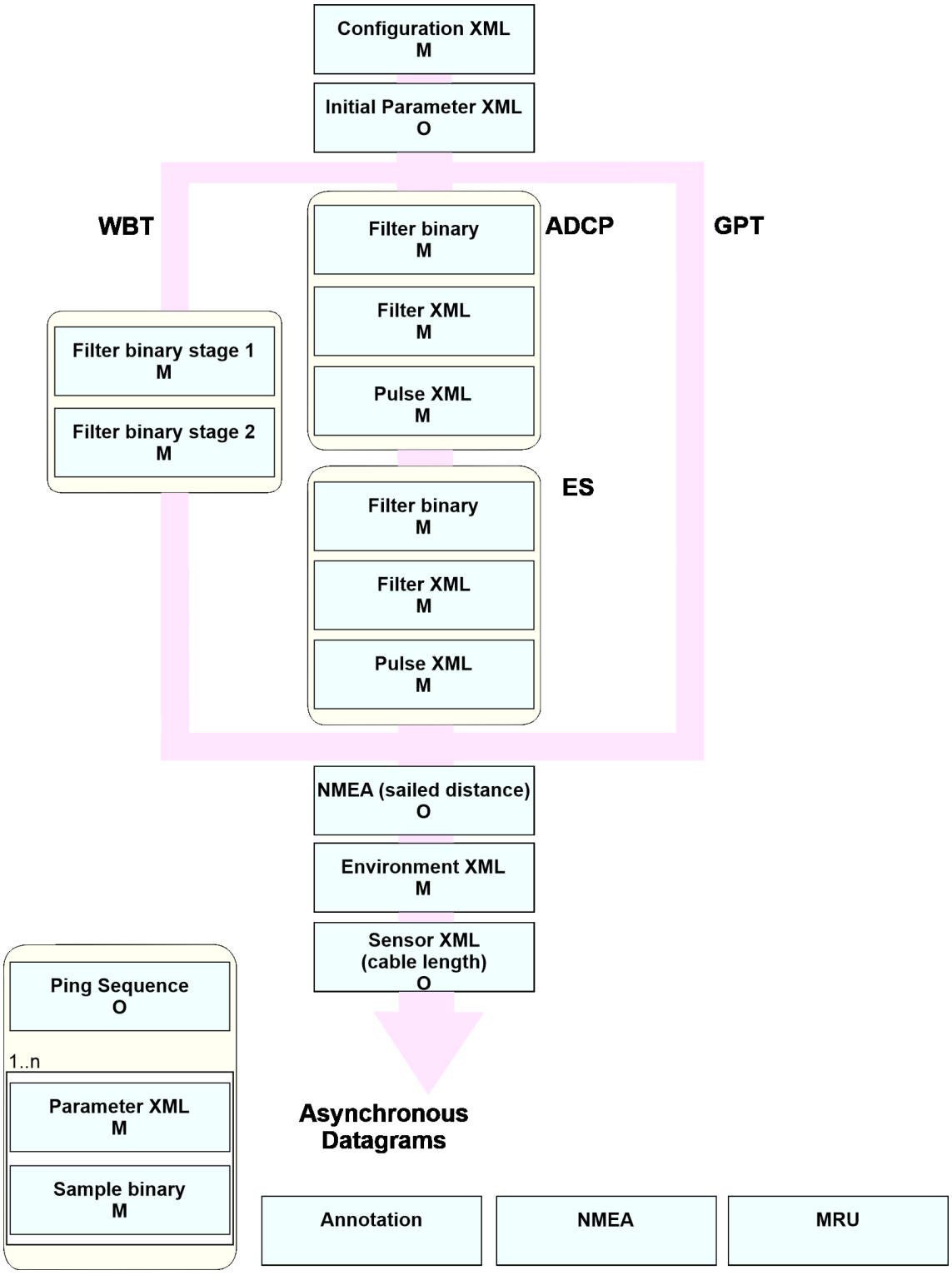
Topics

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

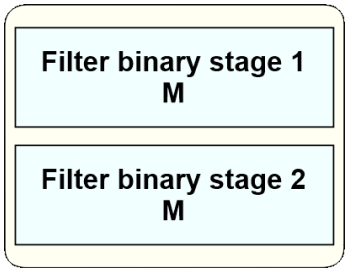
Transceiver configuration details

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".



- **Filter binary datagram**

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

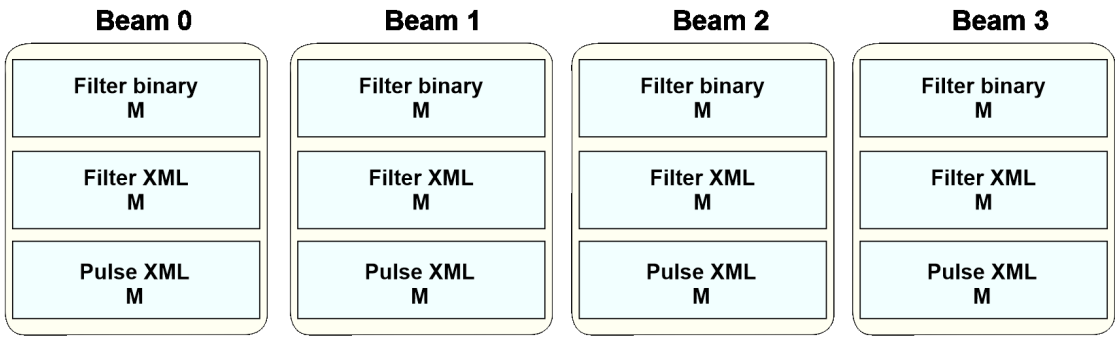
  The Filter binary datagrams contains filter coefficients used for filtering the received signal.

EC150-3C

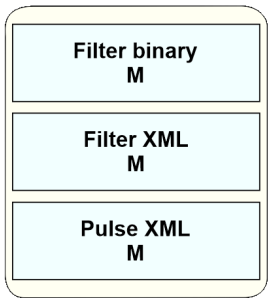
The EC150–3C configuration details include both current profile (ADCP) mode and echo sounder (ES) mode.

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

ADCP



ES



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

Date	Raw format version	EK80 Software version
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

Related topics

Raw data format, page 119
The raw data file format, page 119
Raw data format compatibility, page 123
Using XML datagrams, page 126
Numeric type definition, page 127
Data encapsulation of datagrams, page 125

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 119
The raw data file format, page 119

Raw file datagrams

Topics

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 is specified with this in mind.

Related topics

Raw data format, page 119
Raw file datagrams, page 124
The raw data file format, page 119

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 119
Raw file datagrams, page 124
The raw data file format, page 119

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 119
Raw file datagrams, page 124
The raw data file format, page 119

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 23.6.0.

Updated datagrams

- Configuration XML datagram

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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 119
Raw file datagrams, page 124

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.

Topics

The <Header> tag, page 130

The <ActivePingMode> tag, page 131

The <Transceivers> tag, page 131

The <Transducers> tag, page 136

The <ConfiguredSensors> tag, page 137

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

Related topics

Raw data format, page 119
Raw file datagrams, page 124
Configuration XML datagram, page 129

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

- **1.32**: 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.

<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>
```

The BeamType attribute may have the following values:

- BeamTypeSingle=0,
- BeamTypeSplit=0x1,

- `BeamTypeRef=0x2,`
- `BeamTypeRefB=0x4,`
- `BeamTypeSplit3=0x11,`
- `BeamTypeSplit2=0x21,`
- `BeamTypeSplit3C=0x31,`
- `BeamTypeSplit3CN=0x41,`
- `BeamTypeSplit3CW=0x51,`
- `BeamTypeSplit4B=0x61,`
- `BeamTypeADCPSingle=0x100,`
- `BeamTypeADCPSingle3=0x180,`
- `BeamTypeSinglePassive=0x10`

`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 119
Raw file datagrams, page 124
Configuration XML datagram, page 129

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 119
Raw file datagrams, page 124
Configuration XML datagram, page 129

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 119
Raw file datagrams, page 124
Configuration XML datagram, page 129

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="1491.435067861"
      Temperature="10"
      Latitude="45"
      SoundVelocityProfile="1.000000;1500.000000;1000.000000;1500.000000"
      SoundVelocitySource="Calculated"
      DropKeelOffset="0"
      DropKeelOffsetIsManual="0"
      WaterLevelDraft="0"
      WaterLevelDraftIsManual="0"
      TowedBodyDepth="0"
      TowedBodyDepthlsManual="0"
      >
      <Transducer
            TransducerName="Unknown"
            SoundSpeed="1490"
            />
</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*
- `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

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

  `TowedBodyDepth`, `TowedBodyDepthlsManual`

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

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

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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

- **1.23** New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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

- **1.25**:New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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 is a parameter 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 119
Raw file datagrams, page 124

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.
- `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 seconds.
- `SampleInterval`: The value is given in seconds.
- `TransmitPower`: The value is given in decibels.

- `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`: 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

- **1.23**: New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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`.

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 119
Raw file datagrams, page 124

Motion binary datagram 1

The MRU binary datagram contains information from the Motion Reference Unit (MRU) regarding, heave, pitch and rollThis 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 Maritime for general use.

Record of changes

- **1.23**: New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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 119
Raw file datagrams, page 124

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 119
Raw file datagrams, page 124

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

- **1.23**: New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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

- **1.23**: New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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 is a parameter 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 119
Raw file datagrams, page 124

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 is a parameter 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

- **1.24**: New

Sensor XML datagram

The Sensor XML datagram contains information about cable length for external sensors. This is an XML datagram. The type is `XML0`.

Format

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

```
<Sensor>
     <Sensor Type="CableLength"
     ManualValue="0"
     IsManual="0"
     />
</Sensor>
```

Description

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

- `Sensor Type`: This is the sensor type. The value is given in metres.

Record of changes

- **1.23**: New

Related topics

Raw data format, page 119
Raw file datagrams, page 124

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".

- *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 1.0 and extended with additional data and metadata to store ADCP data.

The echo sounder NetCDF4 information uses SONAR-NetCDF4 version 1.0.

SONAR-NetCDF4 convention version 1.1 has been extended to store split beam echo sounder data. The SONAR-NetCDF4 convention has also been extended with additional data and meta-data to store ADCP data. For the latest update of the convention and the latest approved version please visit:

www.simrad.net/ek80/index.htm

BOT data format

Topics

The BOT data file format

The BOT file format is a proprietary file format designed by Kongsberg Maritime 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 163
The BOT data file format, page 163
Recording parameters, page 164
Simrad BOT depth datagram format, page 164

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 163
The BOT data file format, page 163
Recording parameters, page 164
Simrad BOT depth datagram format, page 164

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

- **1.23**: New

Related topics

BOT data format, page 163
The BOT data file format, page 163
Recording parameters, page 164
Simrad BOT depth datagram format, page 164

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

Topics

About NMEA and standard datagram formats

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

About NMEA and standard datagram formats, page 167
I/O Datagram formats, page 166

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

About NMEA and standard datagram formats, page 167
I/O Datagram formats, page 166

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 Maritime equipment, the `$` character is replaced by the `@` character. The checksum field may then not be in use.

Related topics

About NMEA and standard datagram formats, page 167
I/O Datagram formats, page 166

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

About NMEA and standard datagram formats, page 167
I/O Datagram formats, page 166

NMEA datagram formats

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 in 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 170

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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 170

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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 170

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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 170

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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 (Nautical miles)

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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

NMEA datagram formats, page 170
I/O Datagram formats, page 166

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
NMEA datagram formats, page 170
I/O Datagram formats, page 166

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
NMEA datagram formats, page 170
I/O Datagram formats, page 166

Proprietary datagram formats

Topics

Kongsberg CP1 datagram format

The CP1 Current Profile datagram is a proprietary format created by Kongsberg Maritime. 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 185

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

Proprietary datagram formats, page 185
I/O Datagram formats, page 166

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

Proprietary datagram formats, page 185
I/O Datagram formats, page 166

ATS Annotation datagram format

ATS Annotation is a proprietary datagram format created by Kongsberg Maritime. 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

Proprietary datagram formats, page 185
I/O Datagram formats, page 166

Simrad EK500 Depth datagram

Simrad EK500 Depth is a proprietary datagram format created by Kongsberg Maritime. 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

Proprietary datagram formats, page 185
I/O Datagram formats, page 166

Simrad ITI-FS datagram formats

Topics

Simrad HFB datagram format

Simrad HFB is a proprietary datagram format created by Kongsberg Maritime. 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

Proprietary datagram formats, page 185
Simrad ITI-FS datagram formats, page 190
I/O Datagram formats, page 166

Simrad DBS datagram format

Simrad DBS is a proprietary datagram format created by Kongsberg Maritime 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

Proprietary datagram formats, page 185
Simrad ITI-FS datagram formats, page 190
I/O Datagram formats, page 166

Simrad TDS datagram format

Simrad TDS is a proprietary datagram format created by Kongsberg Maritime 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

Proprietary datagram formats, page 185
Simrad ITI-FS datagram formats, page 190
I/O Datagram formats, page 166

Simrad TPR datagram format

Simrad TPR is a proprietary datagram format created by Kongsberg Maritime. 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

Proprietary datagram formats, page 185
Simrad ITI-FS datagram formats, page 190
I/O Datagram formats, page 166

Simrad TPT datagram format

Simrad TPT is a proprietary datagram format created by Kongsberg Maritime 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

Proprietary datagram formats, page 185
Simrad ITI-FS datagram formats, page 190
I/O Datagram formats, page 166

Simrad PI50 datagrams

Topics

Simrad PSIMDHB datagram format

The proprietary Simrad PSIMDHB datagram format is created by Kongsberg Maritime 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

Proprietary datagram formats, page 185
Simrad PI50 datagrams, page 194
I/O Datagram formats, page 166

Simrad PSIMP,D datagram format

Simrad PSIMP ,D is a proprietary datagram format created by Kongsberg Maritime 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 Maritime.

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

Proprietary datagram formats, page 185
Simrad PI50 datagrams, page 194
I/O Datagram formats, page 166

Simrad PSIMP,F datagram format

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

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

Proprietary datagram formats, page 185
Simrad PI50 datagrams, page 194
I/O Datagram formats, page 166

Kongsberg EM Attitude 3000 datagram format

The Kongsberg EM Attitude 3000 is a proprietary datagram format created by Kongsberg Maritime 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	

Data description	Example	Format	Valid range
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 Maritime. 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

Proprietary datagram formats, page 185
I/O Datagram formats, page 166

KM Binary datagram format

KM Binary is a proprietary datagram format created by Kongsberg Maritime 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

Data description	Unit of measurement	Format	No. of bytes
Height error	m	float	4F
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

Bit	
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 185

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

Third-party datagram and file formats, page 202
I/O Datagram formats, page 166

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

Related topics

Third-party datagram and file formats, page 202
I/O Datagram formats, page 166

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

Related topics

Third-party datagram and file formats, page 202

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

Related topics

Third-party datagram and file formats, page 202
I/O Datagram formats, page 166

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:

$$Roll_{echo\ sounder} = \arcsin [\sin(Roll_{Euler}) \times \cos(Pitch_{Euler})]$$

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.

Related topics

Third-party datagram and file formats, page 202
I/O Datagram formats, page 166

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

- **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 202

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,EHTaaa.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 202

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

Related topics

File formats, page 210

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 mulst include a decimal point.

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

6 The last line is used to teminate the inofrmation. 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 210

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 210

A How to calculate power from power data

Topics Covered in this Appendix

♦ Calculating power from `Power` data recorded with GPT and WBT
♦ Calculating power from `Complex` data recorded for WBT
♦ Calculating power from `Complex` data recorded for EC150–3C

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 119
Sample binary datagram, page 155
Configuration XML datagram, page 129

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:

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

- **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 155
Configuration XML datagram, page 129

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 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 214

Related topics

Sample binary datagram, page 155
Configuration XML datagram, page 129

B How to calculate angle from angle data

Topics Covered in this Appendix

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 119
Sample binary datagram, page 155

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.

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

- **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 155

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 119

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:

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

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 223
Sample binary datagram, page 155
Configuration XML datagram, page 129
The <Header> tag, page 130
The <Transceivers> tag, page 131
The <Transducers> tag, page 136
The <ConfiguredSensors> tag, page 137

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.

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

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

- **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 223
Sample binary datagram, page 155
Calculating angle from Angle data, page 216
Configuration XML datagram, page 129
The <Header> tag, page 130
The <Transceivers> tag, page 131
The <Transducers> tag, page 136
The <ConfiguredSensors> tag, page 137

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.

$$u_{x'} = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{x_2 y_1 - x_1 y_2}{x_1 x_2 + y_1 y_2}\right)\right)$$
$$u_{y'} = \arcsin\left(\frac{1}{\Lambda}\arctan 2\left(\frac{x_4 y_3 - x_3 y_4}{x_3 x_4 + y_3 y_4}\right)\right)$$

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 223
Sample binary datagram, page 155
Configuration XML datagram, page 129
The <Header> tag, page 130
The <Transceivers> tag, page 131
The <Transducers> tag, page 136
The <ConfiguredSensors> tag, page 137

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 155

C Calculating transducer impedance with WBT

Topics Covered in this Appendix

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 (RMS 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 155

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

E

F

G

H

I

K

L

M

N

O

P

R

S

T

V

W

X

Z

Interface Specifications
Kongsberg EK80