FFI Forsvarets forskningsinstitutt

FFI-NOTAT

Eksternnotat 23/02442

LybinTCPServer 7.0.5 - interface description

Author
Elin Margrethe Bøhler
Prosjektnummer 549701
18 December 2023

Approvers
Roald Otnes, *Research Manager*; Trygve Sparr, *Research Director* .
The document is electronically approved and therefore has no handwritten signature.

Keywords
Undervannsakustikk, Sonar, LYBIN, Programvare, Grensesnitt

Summary
LYBIN is a robust, user friendly and fast acoustic ray-trace simulator. A broad set of parameters is used to accurately calculate the probability of detecting objects in a given area under water with the use of sonar technology. LYBIN can be used both with a graphical user interface and as a stand-alone calculation kernel. The stand-alone calculation kernel is available in two different implementations: LybinCom and LybinTCPServer. This FFI note describes the interface of LybinTCPServer 7.0.5.

Contents

1 Introduction

LYBIN [1], [2] is a well established and frequently used sonar prediction tool owned by the Norwegian Defence Materiel Agency (NDMA) and FFI. It is in operative use by the Norwegian Navy and in a number of other nations, and has been modified and improved for this purpose for more than 30 years. FFI has been responsible for testing, evaluation and development of LYBIN since 2000 and has been responsible for commercial sale and support since 2009.

LYBIN is a robust, user friendly and fast acoustic ray-trace simulator. A broad set of parameters is used to accurately calculate the probability of detecting objects in a given area under water with the use of sonar technology. As this probability changes with environmental properties, LYBIN rapidly calculates the sonar coverage.

Several thousand acoustic rays are simulated traversing the water volume. Upon hitting the sea surface and sea bed, the rays are reflected and exposed to loss mechanisms. Losses in the water volume itself due to thermal absorption are accounted for. LYBIN estimates the probability of detection for a given target, based on target echo strength, the calculated transmission loss, reverberation and noise. Both active and passive sonar systems can be simulated.

LYBIN can be used both with a graphical user interface [3] and as a stand-alone calculation kernel. This duality enables LYBIN to interact with other applications, such as mathematical models, web services, geographic information systems, and more. The software is integrated in combat system software, tactical decision aids and tactical trainers. LYBIN has become an important tool in both planning and evaluation of maritime operations [4],[5].

The stand-alone calculation kernel is available in two different implementations; LybinCom and LybinTCPServer. LybinCom [6] is implemented as a Microsoft COM [7] module for the Windows platform. LybinTCPServer is based on Apache Thrift [8], using TCP/IP remote procedure calls. LybinTCPServer can be built for both Windows and Linux platforms and used from multiple programming languages.

This document describes the interface of LybinTCPServer 7.0.5. The three following chapters describe the separate parts of the interface. Chapter 2 gives a description of all the input parameters that can be used in the simulations. Chapter 3 gives a description of how to initiate a sonar performance calculation and Chapter 4 gives a description of all the calculation results available from the calculation. To make it easier for the reader, we have included hyperlinks in the text. The hyperlink is indicated with blue, underlined text, and will direct the reader to the description of the mentioned parameter, class or type.

The Apache Thrift technology enables LybinTCPServer to be used from many different programming languages. In Chapter 2 we have included some parts of code written in C#. This is meant as examples, not limitations. In Appendix A at the end of this document, we have included more complete code examples both for C# and for Python.

2 LYBIN model data

The LybinModelData class contains all the parameters to be used in a simulation: the environment, the platform and all the parameters controlling the acoustic calculations. All the parameters in LybinModelData are listed in *Table 2.1*.

LYBIN has two levels of precision, called cells and steps. Cells describe the precision of the output results grid. Steps describe the precision of the internal calculation grid. The relation between cells and steps is by default so that the number of range steps is 10 times the number of range cells and the number of depth steps is 20 times the number of depth cells. To avoid too large steps, there is a maximum range step size of 50 meters and a maximum depth step size of 5 meters. If the maximum size is exceeded, additional steps are added.

The parameters TypeOfRevNoiseCalculation, UseMeasuredBeamPattern, UseMeasuredBottomLoss, UseMeasuredHorizontalBeamWidth, UseMeasuredPassiveProsessingGain, UseMeasuredSurfaceBackScatter, UseMeasuredSurfaceLoss, UseMeasuredSurfaceReflectionAngles, UseMeasuredTargetStrength and UseRayleighBottomLoss can make LybinTCPServer use certain datasets instead of predefined default values. In order to follow these demands, the spesified datasets must be sent into LybinTCPServer. If LybinTCPServer cannot find these datasets, the switches will be set back to default values.

Parameter	Type	Default value	Unit
DepthCells *Number of depth cells in the calculation output.*	Integer	50	
DepthCellSize *Size of the depth cells in the calculation output.*	Double	6	m
DepthScale *Maximum depth in the calculation.*	Double	300	m
DepthSteps *Number of depth steps to be used during the calculation.*	Integer	1000	
DepthStepSize	Double	0.3	m

Parameter	Type	Default value	Unit
Size of the depth steps to be used during the calculation.			
Environment *All the environmental data to be used in the calculation.*	Environment		
ImpulseResponseCalculation *Switch to control whether to calculate impulse response or not.*	Boolean	false	
ImpulseResponseDepth *The depth that the impulse response will be calculated from.*	Double	0	m
ImpulseResponseWindowHeight *The heigth of the window that the impulse response will be calculated from.*	Double	80	m
MaxBorderHits *Maximum number of boundary hits (sea or bottom) allowed before a ray is terminated.*	Integer	5000	
NoiseCalculation *Switch to control whether to calculate the noise or not.*	Boolean	true	
PassiveCalculation *Switch to control whether to perform calculations for passive or active sonar.*	Boolean	false	
Platform *All the platform data to be used in the calculation.*	Platform		
RangeCells *Number of range cells in the calculation output.*	Integer	50	
RangeCellSize *Size of the range cells in the calculation output.*	Double	200	m

Parameter	Type	Default value	Unit
RangeScale *Maximum range in the calculation.*	Double	10000	m
RangeSteps *Number of range steps to be used during the calculation.*	Integer	500	
RangeStepSize *Size of the range steps to be used during the calculation.*	Double	20	m
SignalExcessConstant *Parameter affecting the relation between signal excess and probability of detection.*	Double	3	
TerminationIntensity *Each ray is terminated when its intensity falls below this value.*	Double	1E-16	
TravelTimeAngleRes *The distance in degrees between the start angles of the rays to be used in the travel time calculation.*	Double	1	Deg
DoTravelTimeCalculation *Switch to control whether to calculate travel time or not.*	Boolean	false	
TRLRays *Number of rays to be used in the transmission loss calculation.*	Integer	1000	
TypeOfRevNoiseCalculation *Enumerator used to control how the calculation of reverberation is performed:* *0: Calculate bottom reverberation from bottom types* *1: Calculate bottom reverberation from back scatter values* *2: Use measured reverberation and noise data* *3: Use Lamberts law to calculate bottom reverberation*	Enum	0	

Parameter	Type	Default value	Unit
UseMeasuredBeamPattern *Tells the model to use measured beam pattern.*	Boolean	false	
UseMeasuredBottomLoss *Tells the model to use measured bottom loss.* *If UseRayleighBottomLoss = true, it will overrule UseMeasuredBottomLoss.*	Boolean	false	
UseMeasuredHorizontalBeamWidth *Tells the model to use the input parameter BeamWidthHorizontal instead of calculating the horizontal beam.*	Boolean	false	
UseMeasuredPassiveProcessingGain *Tells whether to use the input parameter PassiveProcessingGain instead of calculating the passive processing gain.*	Boolean	false	
UseMeasuredSurfaceBackScatter *Tells the model to use measured back scatter instead of calculating it.*	Boolean	false	
UseMeasuredSurfaceLoss *Tells the model to use measured surface loss instead of calculating it.*	Boolean	false	
UseSurfaceReflectionAngles *Tells the model to use input reflection angles instead of calculating them.*	Boolean	false	
UseMeasuredTargetStrength *Tells the model to use measured target strengt.*	Boolean	false	

Parameter	Type	Default value	Unit
UseRayleighBottomLoss *Tells the model to calculate bottom loss according to the Rayleigh bottom loss algorithms.* *If UseRayleighBottomLoss = true, it will overrule UseMeasuredBottomLoss.*	Boolean	false	
UseWaveHeight *Tells the model to use wave height instead of wind speed.*	Boolean	false	
VisualRayTraceCalculation *Switch to control whether to calculate a ray trace plot for visualisation or not.*	Boolean	false	
VisualBottomHits *Number of bottom hits allowed in the visual ray trace plot.*	Integer	1	
VisualNumRays *Number of rays in the visual ray trace plot.*	Integer	50	
VisualSurfaceHits *Number of surface hits alloved in the visual ray trace plot.*	Integer	2	

Table 2.1 Parameters in the LybinModelData class.

2.1 Environment

The environment class contains all the environmental data as listed in *Table 2.2.*

LybinTCPServer is able to handle range dependent environments. In LybinTCPServer, range dependent environmental data are specified for certain range intervals from the sonar. We call such a dataset, with start and stop ranges related to a value (or sets of values), a range dependent object. A range dependent object can contain one or more values with their range of validity. The structure of range dependent objects with start and stop range is shown in *Figure 2.1* . The maximum number of range dependent values are only limited by the given calculation accuracy.

When the environmental properties are entered for a discrete set of locations (ranges), LybinTCPServer will create values at intermediate ranges using interpolation. If no environmental descriptions are given at zero range, LybinTCPServer will substitute the data for the nearest range available, likewise, if data at maximum range are missing.

Parameter	Type
BottomBackScatter	List<StartStopDoubleList>
BottomLoss	List<StartStopDoubleList>
BottomProfile	List<BottomProfileSample>
BottomType	List<StartStopSampleDouble>
LambertsCoefficient	List<StartStopSampleDouble>
Ocean	Ocean
RayleighBottomLoss	RayleighBottomLoss
ReverberationAndNoise	List<ReverberationAndNoiseSample>
SoundSpeed	List<SoundSpeedProfile>
SurfaceBackScatter	List<StartStopDoubleList>
SurfaceLoss	List<StartStopDoubleList>
SurfaceReflectionAngle	List<StartStopSampleDouble>
TargetStrength	List<StartStopDoubleList>
VolumeBackScatter	List<VolumeBackScatterProfile>
WaveHeight	List<StartStopSampleDouble>
WindSpeed	List<StartStopSampleDouble>

Table 2.2 The environment class holds all the environment data.

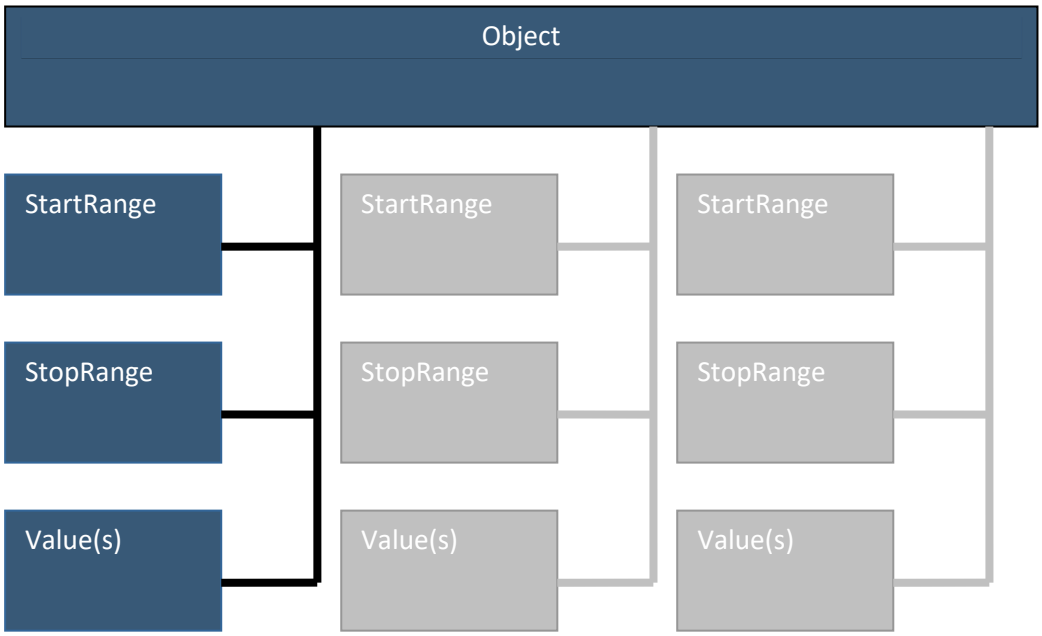

Figure 2.1 Schematic description of a range dependent object with start and stop parameters.

The start and stop functionality provides great flexibility in defining the environmental range dependent properties. By setting start and stop to the same range, the values will be considered to belong to a point in space, and LybinTCPServer will use interpolation to produce data for intermediate range points. The start and stop functionality might be utilized to illustrate meteorological or oceanographic fronts, entering values with finite ranges of validity to each side of the front, and separating the sets by any small distance, across which the conditions will change as abruptly as the user intends. In between these two extreme choices, all combinations of these are possible to use.

In addition to some classes designed for a certain type of environmental data, there are three more generic data classes. They are used for environmental datasets with similar structure. StartStopSampleDouble and StartStopDoubleList store range dependent data and DoubleSample stores angle dependent data. Each of the three classes are listed with parameters, types, default values and units in *Table 2.3*.

The StartStopSampleDouble contains start, stop and value. Both start and stop always has the default 0 and unit m. The default and unit for Value varies with type of environmental data.

The StartStopDoubleList contains start, stop and Samples. Both start and stop always has the default 0 and unit m. Samples is a list of DoubleSamples that contains the two parameters Data and Key. Data has the default value 0 and unit dB. Key has default value 0 and the unit degrees.

The BottomProfile and the ReverberationAndNoiseMeasurements do not have the start-stop functionality. These datasets are not likely to have constant values over range. Both BottomProfile and the ReverberationAndNoiseMeasurements are to be inserted into LybinTCPServer as single values with corresponding range. The number of data points in each dataset is optional.

Class	Parameter	Type	Default value	Unit
DoubleSample	**Data**	Double	0	dB
	Key	Double	0	deg
StartStopSampleDouble	**Start**	Integer	0	m
	Stop	Integer	0	m
	Value	Double	-	-
StartStopDoubleList	**Start**	Integer	0	m
	Stop	Integer	0	m
	Samples	List<DoubleSample>	-	-

Table 2.3 Different types of data classes used to store environmental data. StartStopSampleDouble and StartStopDoubleList store range dependent data. DoubleSample stores angle dependent data.

2.1.1 Bottom back scatter

Bottom back scatter is the fraction of energy that is scattered back towards the receiver when a ray hits the sea bottom. A dataset representing bottom back scattering coefficients is entered into LybinTCPServer in tabular form, giving backscattering coefficients (in dB) for a set of

grazing angles. Based on the tabulated values, LybinTCPServer interpolates between the given values. The back scattering coefficients are given as dB per square meter.

Bottom back scatter is one of four possible options to calculate bottom reverberation. LybinTCPServer will only use the bottom back scatter values given if the TypeOfRevNoiseCalculation parameter in LybinModelData class is set to 1: USE_BOTTOM_BACKSCATTER.

BottomBackScatter is a StartStopDoubleList as listed in *Table 2.3*. A C# example where a start stop double list containing range dependent bottom back scatter values are added to the BottomBackScatter class is seen in *Figure 2.2*.

```
var bb1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 20    //angle in degrees
};

var bb2 = new DoubleSample
{
    Data = 40,  //value in dB
    Key = 25    //angle in degrees
};

var bbs = new StartStopDoubleList
{
    Start = 0,
    Stop = 7000,
    Samples = new List<DoubleSample> { bb1, bb2 }
};

data.Environment.BottomBackScatter.Add(bbs);
data.TypeOfRevNoiseCalculation =
    RevNoiseCalculationType.USE_BOTTOM_BACKSCATTER;
```

Figure 2.2 C# code example: A StartStopDoubleList containing range dependent bottom back scatter values is added to the BottomBackScatter class.

2.1.2 Bottom loss

Bottom loss is the fraction of energy that is lost after the sound has been reflected from the ocean bottom, usually expressed in dB. The bottom loss is also referred to as *forward scattering* in underwater acoustic terminology. A dataset representing bottom loss is entered into LybinTCPServer in tabular form, giving bottom loss (in dB) for a set of grazing angles. Based on the tabulated values, LybinTCPServer interpolates between tabulated values to create loss values for grazing angles in between the given angles.

The parameter UseMeasuredBottomLoss tells LybinTCPServer to use BottomLossTable instead of calculating the bottom loss. If UseRayleighBottomLoss is set to true, UseMeasuredBottomLoss will be ignored. UseRayleighBottomLoss must always be set to false and UseMeasuredBottomLoss to true if one wants to use predefined bottom loss values in LybinTCPServer. Both these parameters can be found in the LybinModelData class.

BottomLoss is a StartStopDoubleList as listed *in Table 2.3*. A C# example where a start stop double list containing range dependent bottom loss values is added to the BottomLoss class is seen in *Figure 2.3*.

```
var blv1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 5     //angle in degrees
};

var blv2 = new DoubleSample
{
    Data = 33,  //value in dB
    Key = 8     //angle in degrees
};

var bl1 = new StartStopDoubleList
{
    Start = 0,
    Stop = 4000,
    Samples = new List<DoubleSample> { blv1, blv2 }
};

data.Environment.BottomLoss.Add(bl1);
data.UseMeasuredBottomLoss = true;
```

Figure 2.3 C# code example: A StartStopDoubleList containing range dependent bottom loss values is added to the BottomLoss class.

2.1.3 Bottom profile

The BottomProfile consist of bottom profile samples containing range and depth values as listed in *Table 2.4*. The samples can be added to the bottom profile as seen in the C# example shown in *Figure 2.4.*

Class	Parameter	Type	Default value	Unit
BottomProfileSample	**Depth**	Double	200	m
	Range	Double	0	m

Table 2.4 The BottomProfileSample contains range and depth.

```
var bps1 = new BottomProfileSample
{
    Depth = 220,
    Range = 50
};

var bps2 = new BottomProfileSample
{
    Depth = 200,
    Range = 0
};
data.Environment.BottomProfile.Add(bps1);
data.Environment.BottomProfile.Add(bps2);
```

Figure 2.4 C# code example: A BottomProfileSample containing range and depth values is added to the BottomProfile class.

2.1.4 Bottom type

The geo-acoustic properties of the bottom are coded by a single parameter in LybinTCPServer. Bottom types ranging from 1 to 9, where 1 represents a hard, rock type of bottom with low bottom reflection loss, while 9 represents a soft bottom with a high reflection loss. In addition,

bottom types 0 and 10 have been added, representing *lossless* and *fully absorbing* bottoms, respectively.

Bottom type is one of three options for modelling the bottom loss. Bottom type is the default choice if both UseMeasuredBottomLoss and UseRayleighBottomLoss are set to false, which also are their default setting. Both these parameters can be found in the LybinModelData class.

Bottom type is the default of the four possible options to calculate bottom reverberation. LybinTCPServer will use the given bottom type when the TypeOfRevNoiseCalculation parameter in LybinModelData class is set to 0: USE_MODELL_CALC_ALL.

A range dependent bottom type sample can be added to the BottomType class as seen in the C# example in *Figure 2.5.* Each range dependent bottom type is given as a StartStopSampleDouble as listed in *Table 2.3.* The default bottom type value is 2.

```
var bt1 = new StartStopSampleDouble
{
    Start = 1200,
    Stop = 7700,
    Value = 3
};

var bt2 = new StartStopSampleDouble
{
    Start = 0,
    Stop = 0,
    Value = 4
};

data.Environment.BottomType.Add(bt1);
data.Environment.BottomType.Add(bt2);
data.TypeOfRevNoiseCalculation = RevNoiseCalculationType.USE_MODELL_CALC_ALL;
```

Figure 2.5 C# code example: A StartStopSampleDouble containing range dependent bottom type data is added to the BottomType class.

2.1.5 Lamberts coefficient

Lamberts rule is one of four possible options to calculate bottom reverberation. According to Lamberts rule, the back scattering coefficient is given by:

$$\sigma(\theta) = \mu \sin^2 \theta \qquad (2.1)$$

Where σ is the back scattering coefficient, θ is the incident grazing angle and μ is the Lamberts coefficient.

The input parameter LambertsCoefficient is range dependent, and needs corresponding start and stop values. If LambertsCoefficient is to be used, the parameter TypeOfRevNoiseCalculation has to be set to 3: USE_LAMBERT_BACKSCATTER, in order to use Lamberts rule in the calculation of the bottom reverberation. The parameter TypeOfRevNoiseCalculation can be found in the LybinModelData class.

The range dependent Lamberts coefficient can be added to the LambertCoefficient class as seen in the C# example in *Figure 2.6.* Each coefficient is given as a StartStopSampleDouble as listed in *Table 2.3*. The default LambertsCoefficient value is 0 dB.

```
var lc1 = new StartStopSampleDouble
{
    Start = 500,
    Stop = 8000,
    Value = 22
};

var lc2 = new StartStopSampleDouble
{
    Start = 8000,
    Stop = 12000,
    Value = 23
};

var lcList = new List<StartStopSampleDouble> { lc1, lc2 };
data.Environment.LambertsCoefficient.AddRange(lcList);
data.TypeOfRevNoiseCalculation =
    RevNoiseCalculationType.USE_LAMBERT_BACKSCATTER;
```

Figure 2.6 C# code example: Two StartStopSampleDoubles containing a range dependent Lamberts coefficient are added to the LambertsCorfficient class.

2.1.6 Ocean

The parameters in the ocean class represent the ocean environment and targets within the sea. All the parameters in the ocean class are listed in *Table 2.5*.

Both Ambient noise and target strength can either be given as a fixed parameter, or it can be calculated from the given environmental input. Which one of these alternatives to be used is decided by the parameters NoiseCalculation and UseMeasuredTargetStrength in LybinModelData.

Parameter	Type	Default value	Unit
AmbientNoiseLevel *Noise from ambient sources.*	Double	50	dB
PH *pH level in the sea water.*	Double	8	
PrecipitationType *Type of precipitation in the area.* *0: No precipitation* *1: Light rain* *2: Heavy rain* *3: Hail* *4: Snow*	Enum	0	
ReverberationZone *The reverberation zone that the target is within, relative to the ship. This parameter is only applicable to CW-pulses.* *0: MainLobe* *1: Typical* *2: NoReverb*	Enum	1	
ShipDensity *Density of ship traffic in the area of the calculation. The ship density can vary from 1 (low) to 7 (high).*	Double	4	
SurfaceScatterFlag	Boolean	true	

Parameter	Type	Default value	Unit
Tells the model if reflected ray angles will be modified in order to simulate rough sea scattering or reflected specularly, as from a perfectly smooth surface.			
TargetAspectAngle *Aspect angle of target.*	Double	0	Deg
TargetCourse *Course of target.*	Double	0	Deg
TargetSpeed *Speed of target.*	Double	10	m/s
TargetStrength *Target echo strength.*	Double	10	dB

Table 2.5 Parameters in the Ocean class.

2.1.7 Rayleigh bottom loss

In order to calculate the bottom loss more accurately, a Rayleigh bottom loss model is included. The Rayleigh bottom loss is based on the physical parameters bottom attenuation, bottom sound speed and density ratio. In order to relate these bottom parameters to other bottom models, the sound speed in the water at bottom depth is assumed to be 1500 m/s. This sound speed is only used in the calculation of bottom loss, and will not influence any other part of the model. The Rayleigh bottom loss is not range dependent. The parameters in the RayleighBottomLoss class are listed in *Table 2.6*. A C# example where Rayleigh bottom loss values are added to the RayleighBottomLoss class is shown in *Figure 2.7*.

In order to make LybinTCPServer calculate and use Rayleigh bottom loss, the UseRayleighBottomLoss parameter in LybinModelData class must be set to true. This parameter will overrule the parameter UseMeasuredBottomLoss if there is any conflict between the settings of the two.

Class	Parameter	Type	Default value	Unit
RayleighBottomLoss	**BottomAttenuation**	Double	0,5	dB/wavelength
	BottomSoundSpeeed	Double	1700	m/s
	DensityRatio	Double	2	

Table 2.6 Parameters in the RayleighBottomLoss class.

```
var rbl = new RayleighBottomLoss
{
    BottomAttenuation = 1,
    BottomSoundSpeed = 1,
    DensityRatio = 1
};

data.Environment.RaylieghBottomLoss = rbl;
data.UseRayleighBottomLoss = true;
```

Figure 2.7 C# code example: Rayleigh bottom loss values are added to the RayleighBottomLoss class.

2.1.8 Reverberation and noise measurements

The ReverberationAndNoiseMeasurements can consist of any number of measurements with corresponding ranges. To find values for the ranges not given as measurements, LybinTCPServer uses linear interpolation.

Reverberation and noise measurements are an optional choice where one uses measured values instead of letting LybinTCPServer estimate reverberation and noise. LybinTCPServer will only use the reverberation and noise measurements values given if the TypeOfRevNoiseCalculation parameter in LybinModelData class is set to 2: USE_MEASURED_REV_NOISE.

The ReverberationAndNoiseSample can consist of any reverberation and noise samples containing range and depth values as listed in *Table 2.7*. The samples can be added to the ReverberationAndNoise class as seen in the C# example in *Figure 2.8*.

Class	Parameter	Type	Default value	Unit
ReverberationAndNoiseSample	**Depth**	Double	0	m
	Scatter	Double	80	dB

Table 2.7 The ReverberationAndNoiseSample contains range and depth.

```
var rns1 = new ReverberationAndNoiseSample
{
    Value = 33,
    Range = 100
};

var rns2 = new ReverberationAndNoiseSample
{
    Value = 39,
    Range = 1000
};

var rns3 = new ReverberationAndNoiseSample
{
    Value = 80,
    Range = 0
};

var rnsSamples = new List<ReverberationAndNoiseSample> {rns1, rns2, rns3};
data.Environment.ReverberationAndNoise.AddRange(rnsSamples);
data.TypeOfRevNoiseCalculation = RevNoiseCalculationType.USE_MEASURED_REV_NOISE;
```

Figure 2.8 C# code example: ReverberationAndNoise values are added to the ReverberationAndNoise class.

2.1.9 Sound speed

The sound speed is a function of both range and depth. Since the sound speed is most often measured as depth dependent profiles, the SoundSpeed class can contain multiple sound speed profiles, representative of different ranges. The sound speed profiles contain sound speed samples holding the parameters temperature, salinity and sound speed for a given set of depths, as listed in *Table 2.8*.

One or more sound speed profiles can be added to the SoundSpeed class as seen in the C# example in *Figure 2.9.*

Class	Parameter	Type	Default value	Unit
SoundSpeedSample	**Depth**	Double	0	m
	SoundSpeed	Double	1480	m/s
	Temperature	Double	7,36	°C
	Salinity	Double	35	parts per thousand
SoundSpeedProfile	**Start**	Integer	0	m
	Stop	Integer	0	m
	Latitude *This parameter is not used in the calculations.*	Double	0	deg N
	Longitude *This parameter is not used in the calculations.*	Double	0	deg E
	SoundSpeedSamples	List <SoundSpeedSample>		

Table 2.8 A sound speed profile contains one or more sound speed samples.

```
var sss1 = new SoundSpeedSample
{
    SoundSpeed = 1480,
    Temp = 6.5,
    Salinity = 0,
    Depth = 0
};

var sss2 = new SoundSpeedSample
{
    SoundSpeed = 1480,
    Temp = 6.5,
    Salinity = 0,
    Depth = 100
};

var ssp = new SoundSpeedProfile
{
    Start = 0,
    Stop = 18288,
    Latitude = 0,
    Longitude = 0,
    ShipLatitude = 0,
    SoundSpeedSamples = new List<SoundSpeedSample> { sss1, sss2 }
};

data.Environment.SoundSpeed.Add(ssp);
```

Figure 2.9 C# code example: sound speed samples are added to the SoundSpeed class.

2.1.10 Surface back scatter

Surface back scatter is the fraction of energy that is scattered back towards the receiver when a ray hits the sea surface. A dataset representing surface back scattering coefficients is entered into LybinTCPServer, giving backscattering coefficients (in dB) for the rays hitting the sea surface. Based on the values, LybinTCPServer interpolates to create backscattering coefficients for the grazing angles. The back scattering coefficients are given as dB per square meter.

Surface back scatter is an optional choice to calculate surface reverberation. LybinTCPServer will only use the surface back scatter values given if the UseMeasuredSurfaceBackScatter parameter in LybinModelData class is set to true.

A StartStopDoubleList containing range dependent surface back scatter values can be added to the SurfaceBackScatter class as listed in *Table 2.3* and shown as a C# example in *Figure 2.10.*

```
var sb1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 20    //angle in degrees
};

var sb2 = new DoubleSample
{
    Data = 40,  //value in dB
    Key = 25    //angle in degrees
};

var sbs = new StartStopDoubleList
{
    Start = 0,
    Stop = 7000,
    Samples = new List<DoubleSample> { sb1, sb2 }
};

data.Environment.SurfaceBackScatter.Add(sbs);
data.UseMeasuredSurfaceBackScatter = true;
```

Figure 2.10 C# code example: surface back scatter samples are added to the SurfaceBackScatter class.

2.1.11 Surface loss

Surface loss is the fraction of energy that is lost after the sound has been reflected from the ocean surface, usually expressed in dB. The surface loss is also referred to as *forward scattering* in underwater acoustic terminology. A dataset representing surface loss is entered into LybinTCPServer, giving surface loss (in dB) for a set of grazing angles. Based on the values, LybinTCPServer interpolates to create loss values for the all grazing angles.

The parameter UseMeasuredSurfaceLoss tells LybinTCPServer to use SurfaceLossTable instead of calculating the surface loss. UseMeasuredSurfaceLoss must be set to true if one wants to use

predefined surface loss values in LybinTCPServer. UseMeasuredSurfaceLoss can be found in the LybinModelData class.

A StartStopDoubleList as listed in *Table 2.3* containing range dependent surface loss values can be added to the SurfaceLoss class as seen in the C# example in *Figure 2.11.*

```
var slv1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 5     //angle in degrees
};

var slv2 = new DoubleSample
{
    Data = 33,  //value in dB
    Key = 8     //angle in degrees
};

var sl1 = new StartStopDoubleList
{
    Start = 0,
    Stop = 4000,
    Samples = new List<DoubleSample> { slv1, slv2 }
};

data.Environment.SurfaceLoss.Add(sl1);
data.UseMeasuredSurfaceLoss = true;
```

Figure 2.11 C# code example: surface loss samples are added to the SurfaceLoss class.

2.1.12 Surface reflection angle

Predefined surface reflection angles can be set as seen in the C# example in *Figure 2.12*. Each surface reflection is given as a StartStopSampleDouble as listed in *Table 2.3.*

Surface reflection angle is an optional parameter that can be used to completely control the surface reflection of each ray in a simulation. If surface reflection angle is to be used, the parameter UseSurfaceReflectionAngles must be set to true. The parameter UseSurfaceReflectionAngles can be found in the LybinModelData class.

```
var sra1 = new StartStopSampleDouble
{
    Start = 0,      //range in meters
    Stop = 0,       //range in meters
    Value = 10      //angle in degrees
};

var sra11 = new StartStopSampleDouble
{
    Start = 9900,   //range in meters
    Stop = 17000,   //range in meters
    Value = 8       //angle in degrees
};

data.Environment.SurfaceReflectionAngle.Add(sra1);
data.Environment.SurfaceReflectionAngle.Add(sra11);
data.UseMeasuredSurfaceReflectionAngles = true;
```

Figure 2.12 C# code example: a surface reflection angle StartStopSampleDouble is added to the SurfaceReflectionAngle class.

2.1.13 Target strength

It is possible to include tables of target strength values. Each table consists of target strength values as a function of aspect angle. The aspect angle can be from 0-359°. If only values less than 180° are given in the table, the target strength values are reflected symmetrically through the longitudinal axis of the target. Each target strength table has a valid frequency range with a given minimum and maximum frequency.

The actual aspect angle to be used in the simulation is given in degrees by the parameter TargetAspectAngle. Whether LybinTCPServer shall find target strength from the table or use the parameter TargetStrength, is given by the parameter UseMeasuredTargeStrength. If UseMeasuredTargeStrength is true, the parameter TargetStrength will be updated with the target strength value that was actually used, found in the table based on frequency and target aspect angle.
A StartStopDoubleList containing frequency dependent target strength values can be added to the TargetStrength class as seen in the C# example in *Figure 2.13* and listed in *Table 2.3*.

```
var tsv1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 5     //angle in degrees
};

var tsv2 = new DoubleSample
{
    Data = 33,  //value in dB
    Key = 8     //angle in degrees
};

var ts1 = new StartStopDoubleList
{
    Start = 0,
    Stop = 4000,
    Samples = new List<DoubleSample> { tsv1, tsv2 }
};

data.Environment.TargetStrength.Add(ts1);
data.UseMeasuredTargetStrength = true;
```

Figure 2.13 C# code example: two target strength samples are added to the TargetStrength class.

2.1.14 Volume back scatter

Volume back scatter is the fraction of energy scattered back towards the receiver from the sea volume. Scattering elements in the sea volume can be particles or organic life, like plankton, fish or sea mammals. The volume back scatterers are not distributed uniformly in the sea, and may vary considerably as a function of depth, range and time of the day. In LybinTCPServer, the volume back scatter is given as a profile of back scattering coefficients as a function of depth. Scatter values for the depths between data points are calculated using linear interpolation. The influence region of each profile is determined from the corresponding start range and stop range values.

The volume back scatter profiles contain volume back scatter samples, as listed in *Table 2.9.*

One or more volume back scatter profiles can be added using the function Add, as seen in the C# example in *Figure 2.14.*

Class	Parameter	Type	Default value	Unit
Volume BackScatter Sample	**Depth**	Double	0	m
	Scatter	Double	80	dB
Volume BackScatter Profile	**Start**	Integer	0	m
	Stop	Integer	0	m
	Latitude *This parameter is optional and not used in the simulation.*	Double	0	deg N
	Longitude *This parameter is optional and not used in the simulation.*	Double	0	deg E
	VolumeBackScatter Samples	List<VolumeBackScatter Sample >		

Table 2.9 A VolumeBackScatterProfile contains one or more VolumeBackScatterSamples.

```
var vbs1 = new VolumeBackScatterSample
{
    Depth = 18,
    Scatter = -80
};

var vbs2 = new VolumeBackScatterSample
{
    Depth = 0,
    Scatter = -92
};

var vbs = new VolumeBackScatterProfile
{
    Start = 0,
    Stop = 0,
    VolumeBackScatterSamples = new List<VolumeBackScatterSample> { vbs1, vbs2 }
};

data.Environment.VolumeBackScatter.Add(vbs);
```

Figure 2.14 C# code example: two volume back scatter samples are added to the VolumeBackScatter class.

2.1.15 Wave height

The WaveHeight consists of wave height StartStopSsampleDoubles containing start, stop and height values as listed in *Table 2.3*. All values are in meters. The samples can be added to the WaveHeight class using the Add function as seen in the C# example in *Figure 2.15*.

Wave height is an optional parameter to wind speed. If wave height is to be used, the parameter UseWaveHeight found in the LybinModelData class must be set to true.

```
var wh1 = new StartStopSampleDouble
{
    Start = 2000,
    Stop = 9000,
    Value = 2
};

data.Environment.WaveHeight.Add(wh1);
data.UseWaveHeight = true;
```

Figure 2.15 C# code example: a wave height StartStopSampleDouble is added to the WaveHeight class.

2.1.16 Wind speed

The wind speed consists of wind speed StartStopSampleDoubles containing start, stop and speed values as listed in *Table 2.3.*. Start and stop has the unit meters and the wind speed is measured in m/s. The default wind speed is 0 m/s. The samples can be added to the WindSpeed class as seen in the C# example in *Figure 2.16.*

```
var ws1 = new StartStopSampleDouble
{
    Start = 1000,
    Stop = 5000,
    Value = 10
};

data.Environment.WindSpeed.Add(ws1);
```

Figure 2.16 C# code example: a wind speed StartStopSamplDouble is added to the WindSpeed class.

2.2 Platform

The platform class contains all the relevant information about the platform holding the sonar. The platform is most often a ship but can also be other things like a helicopter or a buoy. The parameters in the platform class are listed in *Table 2.10.*

Parameter	Type	Default value	Unit
Latitude *Actual latitude of platform.*	Double	0	deg
ShipCourse *Platform course relative to north.*	Double	0	deg
SelfNoise *Noise from the platform that holds the sonar.*	Double	50	dB
SelfNoisePassive *Noise from the platform that holds the sonar. To be used in calculations for passive sonars.*	Double	50	dB
Sensor *All the sensor data to be used in the calculation.*	Sensor		
Speed *Speed of the platform that holds the sonar.*	Double	10	Knots

Table 2.10 Parameters in the platform class.

2.2.1 Sensor

The sensor class contains all the relevant information about the sonar. The parameters in the sensor class are listed in *Table 2.11*.

Parameter	Type	Default value	Unit
BeamPatternReceiver *BeamPattern of the receiver.* *If BeamPattern is to be used, the parameter UseMeasuredBeamPattern must be set to true.*	BeamPattern		

Parameter	Type	Default value	Unit
BeamPatternSender *BeamPattern of the sender.* *If BeamPattern is to be used, the parameter UseMeasuredBeamPattern must be set to true.*	BeamPattern		
BeamWidthHorizontal *Horisontal beam width of the sonar.* *If BeamWidthHorizontal is to be used, the parameter UseMeasuredHorizontalBeamWidth must be set to true.*	Double	20	Degrees
BeamWidthReceiver *Vertical beam width of the receiving part of the sonar.*	Double	15	Degrees
BeamWidthTransmitter *Vertical beam width of the transmitting part of the sonar.*	Double	15	Degrees
CalibrationFactor *The parameter is on the interface, but are not yet implemented or used in the calculations.*	Double	0	dB
Depth *Depth of the sonar.*	Double	5	Meters
DetectionThreshold *The strength of the signal relative to the masking level necessary to see an object with the sonar.*	Double	10	dB
DirectivityIndex *The sonars ability to suppress isotropic noise relative to the response in the steering direction.*	Double	20	dB
Frequency *Centre frequency of the sonar.*	Double	7000	Hz
IntegrationTimePassive *Integration time for the passive sonar.*	Double	1	Seconds

Parameter	Type	Default value	Unit
PassiveBandWidth *Band width of the passive sonar.*	Double	100	Hz
PassiveFrequency *Centre frequency of the passive sonar.*	Double	800	Hz
PassiveProcessinGain *Gain of the passive sonar.*	Double	0	dB
Pulse *All the pulse data to be used in the calculation.*	Pulse		
SideLobeReceiver *The suppression of the highest side lobe relative to the centre of the beam for the receiving sonar.*	Double	13	dB
SideLobeTransmitter *The suppression of the highest side lobe relative to the centre of the beam for the transmitting sonar.*	Double	13	dB
SonarTypePassive *Tells whether the passive sonar is narrow- or broadband..* *0: Narrowband* *1: Broadband*	Enumerator	0	
SourceLevel *Source level of the sonar.*	Double	221	dB
SourceLevelPassive *Source level of the possible target in the calculation for passive sonar.*	Double	100	dB
SystemLoss *System loss due to special loss mechanisms in the sea or sonar system, not otherwise accounted for.*	Double	0	dB
TiltReceiver *Tilt of the receiving part of the sonar.*	Double	4	Degrees

Parameter	Type	Default value	Unit
TiltTransmitter *Tilt of the transmitting part of the sonar.*	Double	4	Degrees

Table 2.11 Parameters in the sensor class.

2.2.1.1 BeamPattern

The BeamPattern measurement is an optional choice where one uses values instead of letting LybinTCPServer eastimate the beam pattern. The beam pattern can consist of any number of measurements with corresponding angles. To find values for the ranges not given as measurements, LybinTCPServer uses linear interpolation.

BeamPattern is an optional parameter that can be used to completely control the start intensity of each ray in a simulation. If BeamPattern is to be used, the parameter UseMeasuredBeamPattern must be set to true. The parameter UseMeasuredBeamPattern can be found in the LybinModelData class

Predefined beam patterns can be set as seen in the C# example in *Figure 2.17.* The beam pattern is given as BeamPatternSamples as listed in *Table 2.12*.

Class	Parameter	Type	Default value	Unit
BeamPatternSample	**Angle**	Double	0	Degrees
	Value	Double	0	dB

Table 2.12 The BeamPatternSample contains angle and value.

```
var bp1 = new BeamPatternSample()
{
    Value = 30,
    Angle = 5
};

var bp2 = new BeamPatternSample()
{
    Value = 38,
    Angle = 8
};

data.Platform.Sensor.BeamPatternReceiver.Add(bp1);
data.Platform.Sensor.BeamPatternSender.Add(bp2);
data.UseMeasuredBeamPattern = true;
```

Figure 2.17 C# code example: two BeamPatternSamples are added to BeamPatternReceiver and BeamPatternSender respectively.

2.2.1.2 Pulse

All the information about the pulse is gathered in the pulse class. All the access parameters in the pulse class are listed in *Table 2.13* below.

Parameter	Type	Default value	Unit
EnvelopeFunc *Envelope function of the signal. Currently, only "Hann" is available.*	String	Hann	
FilterBandWidth *Filter bandwidth of the pulse.*	Double	100	Hz
FMBandWidth *Frequency modulation bandwidth of the pulse. Applicable for FM signals only.*	Double	100	Hz

Parameter	Type	Default value	Unit
Form *Pulse type:* *FM: Frequency modulated* *CW: Continuous wave*	String	FM	
Length *Pulse length.*	Double	60	Milliseconds

Table 2.13 Parameters in the pulse class.

3 Initiate calculation

The CalculateLybinModel function initiates a new instance of LybinTCPServer using the modelIndex returned from the function CreateLybinModel that is used to set the model data in the simulation. Both these functions are described in *Table 3.1*.

Function	Type
CalculateLybinModel(int modelIndex) *Start the calculation.*	LybinResults
CreateLybinModel(LybinModelDAta lybinModelData) *Send the model data to LybinTCPServer.*	Integer

Table 3.1 Functions for initiation of calculation.

4 Calculation results

4.1 Functions returning calculation results

The calculation results can be accessed through the functions listed in *Table 4.1*.

Function	Type	Unit
getCalculatedAmbientNoise(int modelIndex) *The ambient noise used in the calculations.*	Double	dB
getBottomReverberation(int modelIndex) *Calculated bottom reverberation values.*	List<double>	dB
getEchoLevel(int modelIndex) *Not yet implemented inside LybinCore. This object will not have any data.*	List<List<double>>	dB
getImpulseResponse(int modelIndex) *Get calculated impulse response.*	List<List<ImpulseResponsePoint>>	
getInterpolatedBottomProfile(int modelIndex) *Get the interpolated bottom profile.*	List<List<double>>	m
GetInterpolatedSoundSpeed(int modelIndex) *Get the smoothed and interpolated sound speed matrix.*	List<List<double>>	m/s
getMaskingLevel(int modelIndex) *Calculated masking level (total reverberation + noise after processing).*	List<double>	dB
getNoiseAfterProcessing(int modelIndex) *Calculated noise after processing.*	Double	dB
getProbabilityOfDetection(int modelIndex) *Calculated probability of detection.*	List<List<double>>	%

Function	Type	Unit
getRayTrace(int modelIndex) *Not implemented inside LybinCore. This object will not have any data.*	List<List<double>>	
getLybinModel(int modelIndex) *The model data used during the calculation.*	LybinModelData	
getSignalExcess(int modelIndex) *Calculated signal excess.*	List<List<double>>	dB
getSurfaceReverberation(int modelIndex) *Calculated surface reverberation.*	List<double>	dB
getTotalReverberation(int modelIndex) *Calculated total reverberation.*	List<double>	dB
getTransmissionLossReceiver(int modelIndex) *Calculated transmission loss from the target to the receiver.*	List<List<double>>	dB
getTransmissionLossTransmitter(int modelIndex) *Calculated transmission loss from the transmitter to the target.*	List<List<double>>	dB
getTravelTime(int modelIndex) *Returns the travel time paths calculated.*	List<List<TravelTimePoint>>	
getVisualRayTrace(int modelIndex) *Returns the visual ray trace paths calculated.*	List<List<VisualRayTracePoint>>	
getVolumeReverberation(int modelIndex) *Calculated volume reverberation.*	List<double>	dB

Table 4.1 Functions returning calculation results.

4.2 Impulseresponse point

All the parameters in the ImpulseResponsePoint class are listed in *Table 4.2.*

Parameter	Type	Unit
LongestTravelTime *Longest travel time.*	Double	s
MaxInitialAngle *Maximum initial ray angle.*	Double	deg
MeanInitialAngle *Mean initial ray angle.*	Double	deg
MinInitialAngle *Minimum initial ray angle.*	Double	deg
My *Mean arrival time – first arrival time.*	Double	s
Phase *Phase identifier.*	Integer	
RayFamilyCode *Ray family identifier. The ray family identifier represents the ray family's travel history, using the letter codes:* *s Surface reflection* *b Bottom reflection* *u Upper turning point* *l Lower turning point*	String	
S *Intensity loss.*	Double	dB
ShortestTravelTime *Shortest travel time.*	Double	s
Sigma *Arrival time standard deviation.*	Double	s

StandardDeviationInitialAngle *Standard deviation of initial ray angle.*	Double	deg

Table 4.2 Parameters in the ImpulseResponsePoint class.

4.3 Traveltime point

All the parameters in the TravelTimePoint class are listed in *Table 4.3*.

Parameter	Type	Unit
InitialAngle *Initial ray angle.*	Double	deg
Range *Range of travel time point.*	Double	m
Depth *Depth of travel time point.*	Double	m
TravelTime *Travel time from start to point.*	Double	s

Table 4.3 Parameters in the TravelTimePoint class.

4.4 Visual raytrace point

All the parameters in the VisualRayTracePoint class are listed in *Table 4.4*.

Parameter	Type	Unit
InitialAngle *Initial ray angle.*	Double	deg
Range *Range of travel time point.*	Double	m

Depth *Depth of travel time point.*	Double	m

Table 4.4 Parameters in the VisualRayTracePoint class.

A Code examples

A.1 C# code example

```
using System.Collections.Generic;
using Lybin.RPC;
using Thrift.Protocol;
using Thrift.Transport;

namespace LybinTCPClient
{
    0 references
    class Program
    {
        0 references
        private static void Main()
        {
            TTransport transport = new TSocket(host: "localhost", port: 9090);
            TProtocol protocol = new TBinaryProtocol(transport);
            var client = new Lybin.RPC.Lybin.Client(protocol);
            transport.Open();
            var data = new LybinModelData
            {
                Environment = new Environment
                {
                    Ocean = new Ocean(),
                    WindSpeed = new List<StartStopSampleDouble>(),
                    WaveHeight = new List<StartStopSampleDouble>(),
                    SoundSpeed = new List<SoundSpeedProfile>(),
                    BottomProfile = new List<BottomProfileSample>(),
                    BottomType = new List<StartStopSampleDouble>(),
                    BottomLoss = new List<StartStopDoubleList>(),
                    RayliehBottomLoss = new RayleighBottomLoss(),
                    BottomBackScatter = new List<StartStopDoubleList>(),
                    LambertsCoefficient = new List<StartStopSampleDouble>(),
                    VolumeBackScatter = new List<VolumeBackScatterProfile>(),
                    ReverberationAndNoise = new List<ReverberationAndNoiseSample>(),
                    SurfaceBackScatter = new List<StartStopDoubleList>(),
                    SurfaceLoss = new List<StartStopDoubleList>(),
                    SurfaceReflectionAngle = new List<StartStopSampleDouble>(),
                    TargetStrength = new List<StartStopDoubleList>()
                }
            };

            var ocean = new Ocean
            {
                PrecipitationNoiseType = PrecipitationType.NO_PRECIPITATION,
                ReverberationZone = ReverberationZone.MAIN_LOBE,
```

```
            AmbientNoiseLevel = 50,
            PH = 7,
            ShipDensity = 1,
            SourceLevelTarget = 1,
            SurfaceScatterFlag = true,
            TargetAspectAngle = 1,
            TargetCourse = 0,
            TargetSpeed = 1,
            TargetStrength = 5
        };
        data.Environment.Ocean = ocean;

        var tsv1 = new DoubleSample
        {
            Data = 30,  //value in dB
            Key = 5     //angle in degrees
        };

        var tsv2 = new DoubleSample
        {
            Data = 33,  //value in dB
            Key = 8     //angle in degrees
        };

        var ts1 = new StartStopDoubleList
        {
            Start = 0,
            Stop = 4000,
            Samples = new List<DoubleSample> { tsv1, tsv2 }
        };

        data.Environment.TargetStrength.Add(ts1);
        //data.UseMeasuredTargetStrength = true;

        var slv1 = new DoubleSample
        {
            Data = 30,  //value in dB
            Key = 5     //angle in degrees
        };

        var slv2 = new DoubleSample
        {
            Data = 33,  //value in dB
            Key = 8     //angle in degrees
        };

        var sl1 = new StartStopDoubleList
        {
            Start = 0,
            Stop = 4000,
            Samples = new List<DoubleSample> { slv1, slv2 }
        };
```

```
data.Environment.SurfaceLoss.Add(sl1);
//data.UseMeasuredSurfaceLoss = true;

var sb1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 20    //angle in degrees
};

var sb2 = new DoubleSample
{
    Data = 40,  //value in dB
    Key = 25    //angle in degrees
};

var sbs = new StartStopDoubleList
{
    Start = 0,
    Stop = 7000,
    Samples = new List<DoubleSample> { sb1, sb2 }
};

data.Environment.SurfaceBackScatter.Add(sbs);
//data.UseMeasuredSurfaceBackScatter = true;

var rns1 = new ReverberationAndNoiseSample
{
    Value = 33,
    Range = 100
};

var rns2 = new ReverberationAndNoiseSample
{
    Value = 39,
    Range = 1000
};

var rns3 = new ReverberationAndNoiseSample
{
    Value = 80,
    Range = 0
};

var rnsSamples =
    new List<ReverberationAndNoiseSample> {rns1, rns2, rns3};
data.Environment.ReverberationAndNoise.AddRange(rnsSamples);
//data.TypeOfRevNoiseCalculation =
    //RevNoiseCalculationType.USE_MEASURED_REV_NOISE;
```

```
var vbs1 = new VolumeBackScatterSample
{
    Depth = 18,
    Scatter = -80
};

var vbs2 = new VolumeBackScatterSample
{
    Depth = 0,
    Scatter = -92
};

var vbs = new VolumeBackScatterProfile
{
    Start = 0,
    Stop = 0,
    VolumeBackScatterSamples =
        new List<VolumeBackScatterSample> { vbs1, vbs2 }
};

data.Environment.VolumeBackScatter.Add(vbs);

var lc1 = new StartStopSampleDouble
{
    Start = 500,
    Stop = 8000,
    Value = 22
};

var lc2 = new StartStopSampleDouble
{
    Start = 8000,
    Stop = 12000,
    Value = 23
};

var lcList = new List<StartStopSampleDouble> { lc1, lc2 };
data.Environment.LambertsCoefficient.AddRange(lcList);
//data.TypeOfRevNoiseCalculation =
    //RevNoiseCalculationType.USE_LAMBERT_BACKSCATTER;

var bb1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 20    //angle in degrees
};

var bb2 = new DoubleSample
{
    Data = 40,  //value in dB
    Key = 25    //angle in degrees
};
```

```
var bbs = new StartStopDoubleList
{
    Start = 0,
    Stop = 7000,
    Samples = new List<DoubleSample> { bb1, bb2 }
};

data.Environment.BottomBackScatter.Add(bbs);
//data.TypeOfRevNoiseCalculation =
    //RevNoiseCalculationType.USE_BOTTOM_BACKSCATTER;


var rbl = new RayleighBottomLoss
{
    BottomAttenuation = 1,
    BottomSoundSpeed = 1,
    DensityRatio = 1
};

data.Environment.RayliеghBottomLoss = rbl;
//data.UseRayleighBottomLoss = true;

var blv1 = new DoubleSample
{
    Data = 30,  //value in dB
    Key = 5     //angle in degrees
};

var blv2 = new DoubleSample
{
    Data = 33,  //value in dB
    Key = 8     //angle in degrees
};

var bl1 = new StartStopDoubleList
{
    Start = 0,
    Stop = 4000,
    Samples = new List<DoubleSample> { blv1, blv2 }
};

data.Environment.BottomLoss.Add(bl1);
//data.UseMeasuredBottomLoss = true;

var bt1 = new StartStopSampleDouble
{
    Start = 1200,
    Stop = 7700,
    Value = 3
};
```

```
            var bt2 = new StartStopSampleDouble
            {
                Start = 0,
                Stop = 0,
                Value = 4
            };

            data.Environment.BottomType.Add(bt1);
            data.Environment.BottomType.Add(bt2);
            data.TypeOfRevNoiseCalculation =
                RevNoiseCalculationType.USE_MODELL_CALC_ALL;

            var bps1 = new BottomProfileSample
            {
                Depth = 220,
                Range = 50
            };

            var bps2 = new BottomProfileSample
            {
                Depth = 200,
                Range = 0
            };
            data.Environment.BottomProfile.Add(bps1);
            data.Environment.BottomProfile.Add(bps2);

            var sss1 = new SoundSpeedSample
            {
                SoundSpeed = 1480,
                Temp = 6.5,
                Salinity = 0,
                Depth = 0
            };

            var sss2 = new SoundSpeedSample
            {
                SoundSpeed = 1480,
                Temp = 6.5,
                Salinity = 0,
                Depth = 100
            };

            var ssp = new SoundSpeedProfile
            {
                Start = 0,
                Stop = 18288,
                Latitude = 0,
                Longitude = 0,
                ShipLatitude = 0,
                SoundSpeedSamples = new List<SoundSpeedSample> { sss1, sss2 }
            };

            data.Environment.SoundSpeed.Add(ssp);
```

```
var ws1 = new StartStopSampleDouble
{
    Start = 1000,
    Stop = 5000,
    Value = 10
};

data.Environment.WindSpeed.Add(ws1);

var ws2 = new StartStopSampleDouble
{
    Start = 0,
    Stop = 0,
    Value = 3.85833
};
data.Environment.WindSpeed.Add(ws2);

var wh1 = new StartStopSampleDouble
{
    Start = 2000,
    Stop = 9000,
    Value = 2
};

data.Environment.WaveHeight.Add(wh1);
//data.UseWaveHeight = true;

var sra1 = new StartStopSampleDouble
{
    Start = 0,      //range in meters
    Stop = 0,       //range in meters
    Value = 10      //angle in degrees
};

var sra11 = new StartStopSampleDouble
{
    Start = 9900,   //range in meters
    Stop = 17000,   //range in meters
    Value = 8       //angle in degrees
};

data.Environment.SurfaceReflectionAngle.Add(sra1);
data.Environment.SurfaceReflectionAngle.Add(sra11);
//data.UseMeasuredSurfaceReflectionAngles = true;

data.DepthCellSize = 2;
data.DepthCells = 50;
data.DepthScale = 100;
data.DepthStepSize = 0.28;
data.DepthSteps = 1000;
```

```
data.ImpulseResponseCalculation = false;
data.ImpulseResponseDepth = 0;
data.ImpulseResponseWindowHeight = 80;
data.NoiseCalculation = false;
data.PassiveCalculation = false;
data.RangeCells = 50;
data.RangeCellSize = 365.76;
data.RangeScale = 18288;
data.RangeSteps = 500;
data.RangeStepSize = 36.576;
data.SignalExcessConstant = 3;
data.TerminationIntensity = 1e-16;
data.TravelTimeAngleRes = 1;
data.DoTravelTimeCalculation = false;
data.TRLRays = 500;
data.UseMeasuredHorizontalBeamWidth = false;
data.UseMeasuredPassiveProcessingGain = false;
data.VisualBottomHits = 1;
data.VisualNumRays = 20;
data.VisualRayTraceCalculation = true;
data.VisualSurfaceHits = 2;

data.Platform = new Platform
{
    Latitude = 0,
    ShipCourse = 0,
    SelfNoise = 50,
    SelfNoisePassive = 50,
    Speed = 0,
    Sensor = new Sensor
    {
        BeamWidthHorizontal = 0,
        BeamWidthReceiver = 15,
        BeamWidthTransmitter = 15,
        CalibrationFactor = 0,
        Depth = 0,
        DetectionThreshold = 10,
        DirectivityIndex = 20,
        Effect = 0,
        Frequency = 5500,
        IntegrationTimePassive = 30000,
        PassiveBandWidth = 100,
        PassiveFrequency = 800,
        PassiveProcessingGain = 0,
        SideLobeReceiver = 16.5,
        SideLobeTransmitter = 16.5,
        SonarTypePassive = SonarType.Narrowband,
        SourceLevelPassive = 150,
        SourceLevel = 188,
        SystemLoss = 0,
        TiltReceiver = 0,
        TiltTransmitter = 0,
```

```
                Pulse = new Pulse
                {
                    EnvelopeFunc = "Hann",
                    FMBandWidth = 500,
                    FilterBandWidth = 1000,
                    Form = "FM",
                    Length = 0.060,
                },

                BeamPatternReceiver = new List<BeamPatternSample>
                {
                    new BeamPatternSample
                    {
                        Value = 5,
                        Angle = 10
                    }
                },
                BeamPatternSender = new List<BeamPatternSample>
                {
                    new BeamPatternSample
                    {
                        Value = 44,
                        Angle = 7
                    },
                    new BeamPatternSample
                    {
                        Value = 55,
                        Angle = 66
                    }
                }
            }
        };

        //data.UseMeasuredBeamPattern = true;

        var index = client.createLybinModel(data);
        var result = client.calculateLybinModel(index);
        var rev = client.getBottomReverberation(index);
        var br = client.getLybinModel(index);
        var echoLevel = client.getEchoLevel(index);
        var impulseResponse = client.getImpulseresponse(index);
        var maskingLevel = client.getMaskingLevel(index);
        var noiseAfterPrepossessing = client.getNoiseAfterProcessing(index);
        var probabilityOfDetection = client.getProbabilityOfDetection(index);
        var rayTrace = client.getRayTrace(index);
        var signalExcess = client.getSignalExcess(index);
        var surfaceReverberation = client.getSurfaceReverberation(index);
        var totalReverberation = client.getTotalReverberation(index);
        var transmissionLossReceiver = client.getTransmisionLossReceiver(index);
        var transmissionLossTransmitter = client.getTransmisionLossTransmitter(index);
        var travelTime = client.getTravelTime(index);
        var visualRayTrace = client.getVisualRayTrace(index);
        var volumeReverberation = client.getVolumeReverberation(index);
        var interpolatedBottomProfile = client.getInterpolatedBottomProfile(index);
        var interpolatedSoundSpeed = client.getInterpolatedSoundSpeed(index);

    }
}
}
```

A.2 Python code example

```
import sys
import glob
import matplotlib.pyplot as plt
import numpy as np
sys.path.append('gen-py')

from Lybin.RPC import Lybin
from Lybin.RPC import ttypes

from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

if __name__ == "__main__":
    # Make socket
    transport = TSocket.TSocket('localhost', 9090)

    # Buffering is critical. Raw sockets are very slow
    transport = TTransport.TBufferedTransport(transport)

    # Wrap in a protocol
    protocol = TBinaryProtocol.TBinaryProtocol(transport)

    client = Lybin.Client(protocol)
    data = ttypes.LybinModelData()

    transport.open()

    data.environment = Lybin.Environment()

    data.environment.ocean = Lybin.Ocean(
        PH=7.0,
        surfaceScatterFlag=False,
        targetStrength=11.0,
        targetSpeed=0.0,
        targetCourse=0.0,
        targetAspectAngle=0.0,
        reverberationZone=2,
        shipDensity=5.0,
        ambientNoiseLevel=55.0,
        sourceLevelTarget=120.0,
        precipitationNoiseType=1)
```

```
data.environment.waveheigth = [
    Lybin.StartStopSampleDouble(start=2000,
                                stop=9000,
                                value = 3.0)]

data.environment.bottomType = [
    Lybin.StartStopSampleDouble(start=0,
                                stop=0,
                                value=3.0)]

data.environment.windSpeed  = [
    Lybin.StartStopSampleDouble(start=0,
                                stop=1000,
                                value=8),
    Lybin.StartStopSampleDouble(start=2000,
                                stop=10000,
                                value=6)]

data.environment.lambertsCoefficient = [
    Lybin.StartStopSampleDouble(start=500,
                                stop=8000,
                                value=22.0),
    Lybin.StartStopSampleDouble(start=8000,
                                stop=12000,
                                value=23.0)]

data.environment.surfaceReflectionAngle = [
    Lybin.StartStopSampleDouble(start=2000,
                                stop=4000,
                                value=2.0),
    Lybin.StartStopSampleDouble(start=4000,
                                stop=6000,
                                value=8.0)]

data.environment.bottomProfile = [
    Lybin.BottomProfileSample(range=0.0,
                              depth=150),
    Lybin.BottomProfileSample(range = 5000,
                              depth = 120)]

data.environment.raylieghBottomLoss = Lybin.RayleighBottomLoss(
    bottomAttenuation=5.0,
    bottomSoundSpeed=1600.0,
    densityRatio=1.0)

data.environment.reverberationAndNoise = [
    Lybin.ReverberationAndNoiseSample(range=0.0,
                                      value=33.0)]
```

```
data.environment.bottomLoss = [
    Lybin.StartStopDoubleList(start=0,
                              stop=0,
                              samples=[
                                  Lybin.DoubleSample(key=5.0,
                                                     data=30.0),
                                  Lybin.DoubleSample(key=8.0,
                                                     data=33.0)])]

data.environment.bottomBackScatter = [
    Lybin.StartStopDoubleList(start=0,
                              stop=5000,
                              samples=[
                                  Lybin.DoubleSample(key=10.0,
                                                     data=30.0),
                                  Lybin.DoubleSample(key=25.0,
                                                     data=40.0)]),
    Lybin.StartStopDoubleList(start=8888,
                              stop=11000,
                              samples=[
                                  Lybin.DoubleSample(key=22.0,
                                                     data=33.0),
                                  Lybin.DoubleSample(key=55.0,
                                                     data=44.0)])]

data.environment.surfaceLoss = [
    Lybin.StartStopDoubleList(start=0,
                              stop=0,
                              samples=[
                                  Lybin.DoubleSample(key=5.0,
                                                     data=30.0),
                                  Lybin.DoubleSample(key=8.0,
                                                     data=33.0)])]

data.environment.surfaceBackScatter = [
    Lybin.StartStopDoubleList(start=0,
                              stop=7000,
                              samples=[
                                  Lybin.DoubleSample(key=20.0,
                                                     data=30.0),
                                  Lybin.DoubleSample(key=25.0,
                                                     data=40.0)]),
    Lybin.StartStopDoubleList(start=8888,
                              stop=11000,
                              samples=[
                                  Lybin.DoubleSample(key=22.0,
                                                     data=33.0),
                                  Lybin.DoubleSample(key=55.0,
                                                     data=44.0)])]
```

```
data.environment.targetStrength = [
    Lybin.StartStopDoubleList(start=0,
                              stop=4000,
                              samples=[
                                  Lybin.DoubleSample(key=5.0,
                                                     data=30.0),
                                  Lybin.DoubleSample(key=8.0,
                                                     data=33.0)]),
    Lybin.StartStopDoubleList(start=5000,
                              stop=7000,
                              samples=[
                                  Lybin.DoubleSample(key=2.0,
                                                     data=20.0),
                                  Lybin.DoubleSample(key=6.0,
                                                     data=66.0)])]

data.environment.soundSpeedProfile = Lybin.SoundSpeedProfile(
    start=0,
    stop=1000,
    latitude=0,
    longitude=0,
    shipLatitude=0,
    soundSpeedSamples=[
        Lybin.SoundSpeedSample(depth=5,
                               soundSpeed=1500,
                               temp=8,
                               salinity=34)])

data.bottomReverberationCalculation = False
data.depthCellSize = 3
data.depthCells = 50
data.depthScale = 150
data.doTravelTimeCalculation = False
data.impulseResponseCalculation = False
data.impulseResponseDepth = 0
data.impulseResponseWindowHeight = 80
data.maxBorderHits = 5000
data.noiseCalculation = True
data.passiveCalculation = False
data.rangeCells = 50
data.rangeCellSize = 200
data.rangeScale = 10000
data.rayTraceCalculation = True
data.signalExcessCalculation = True
data.signalExcessConstant = 3
data.surfaceReverberationCalculation = True
data.terminationIntensity = 1e-16
data.transmissionLossFromTargetCalculation = True
data.transmissionLossToTargetCalculation = True
data.travelTimeAngleRes = 1
data.doTravelTimeCalculation = False
data.TRLRays = 1000
```

```
data.typeOfRevNoiseCalculation = ttypes.RevNoiseCalculationType.USE_MODELL_CALC_ALL
data.useMeasuredBottomLoss = False
data.useMeasuredHorizontalBeamWidth = False
data.useMeasuredPassiveProcessingGain = False
data.useMeasuredSurfaceBackScatter = False
data.useMeasuredSurfaceLoss = False
data.useMeasuredSurfaceReflectionAngles = False
data.useMeasuredTargetStrength = False
data.useRayleighBottomLoss = False
data.useWaveHeight = False
data.visualBottomHits = 1
data.visualNumRays = 50
data.visualRayTraceCalculation = True
data.visualSurfaceHits = 2
data.volumeReverberationCalculation = True

data.platform = Lybin.Platform()
data.platform.latitude = 60.1
data.platform.shipCourse = 0
data.platform.selfNoise = 50
data.platform.selfNoisePassive = 50
data.platform.speed = 10

data.platform.sensor = Lybin.Sensor()
data.platform.sensor.beamWidthReceiver = 15
data.platform.sensor.beamWidthTransmitter = 15
data.platform.sensor.calibrationFactor = 0
data.platform.sensor.depth = 5
data.platform.sensor.detectionThreshold = 10
data.platform.sensor.directivityIndex = 20
data.platform.sensor.frequency = 7000
data.platform.sensor.integrationTimePassive = 1
data.platform.sensor.passiveBandWidth = 100
data.platform.sensor.passiveFrequency = 800
data.platform.sensor.passiveProcessingGain = 0
data.platform.sensor.sideLobeReceiver = 13
data.platform.sensor.sideLobeTransmitter = 13
data.platform.sensor.sourceLevelPassive = 0
data.platform.sensor.sourceLevel = 221
data.platform.sensor.systemLoss = 0
data.platform.sensor.tiltReceiver = 4
data.platform.sensor.tiltTransmitter = 4

data.platform.sensor.pulse = Lybin.Pulse()
data.platform.sensor.pulse.envelopeFunc = "Hann"
data.platform.sensor.pulse.FMBandWidth = 100
data.platform.sensor.pulse.filterBandWidth = 100
data.platform.sensor.pulse.form = "FM"
data.platform.sensor.pulse.length = 60
data.platform.sensor.pulse.processingGainNoise = 0
data.platform.sensor.pulse.processingGainRev = 0
```

```
index = client.createLybinModel(data)
client.calculateLybinModel(index)

transmisionLossTransmitter = client.getTransmisionLossTransmitter(index)
signalExcess = client.getSignalExcess(index)
probabilityOfDetection = client.getProbabilityOfDetection(index)
surfaceReverberation = client.getSurfaceReverberation(index)
volumeReverberation = client.getVolumeReverberation(index)
bottomReverberation = client.getBottomReverberation(index)
totalReverberation = client.getTotalReverberation(index)
noiseAfterProcessing = client.getNoiseAfterProcessing(index)
```

References

1. E. Dombestein, and T. Jenserud, “Improving Underwater Surveillance: LYBIN Sonar performance prediction”, Proceedings of MAST 2010 – Rome, 2010.

2. K.T. Hjelmervik, S. Mjølsnes, E. Dombestein, T. Såstad and J. Wegge, “The acoustic raytrace model Lybin – Descriptions and applications”, UDT 2008, Glasgow, United Kingdom, 2008

3. E. Dombestein, "LYBIN 6.2 2200 - user manual", FFI Rapport 17/00412, 2017.

4. E. Dombestein, S. Mjølsnes, and F. Hermansen, "Visualization of sonar performance within environmental information," in Oceans 2013, Bergen, 2013.

5. E. Dombestein and F. Hermansen, "Integration of Sonar Performance Modelling in Sonar Operator Training, Mission Planning and High Risk Decisions," presented at the MSG-126, Washington DC, USA, 2014.

6. E. Dombestein, "LybinCom 6.2 - description of the binary interface," FFI-Rapport 2014/00511, 2014.

7. Microsoft Component Object Model (COM), https://learn.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal

8. The Apache Thrift software framework, https://thrift.apache.org/

Om FFI
Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan. Med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs formål
Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

FFIs visjon
FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs verdier
Skapende, drivende, vidsynt og ansvarlig.