FFI Forsvarets forskningsinstitutt

FFI-NOTAT

Eksternnotat 24/01031

LybinCom 7.0 – interface description

Author
Elin Margrethe Bøhler
Project number 549701
12 August 2024

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 LybinCom 7.0.

E

Contents

1 Introduction **4**

2 LYBIN model data **5**

2.1 Environment 12

2.1.1 Bottom back scatter 14

2.1.2 Bottom loss 15

2.1.3 Bottom profile 18

2.1.4 Bottom type 18

2.1.5 Lamberts coefficient 19

2.1.6 Ocean 21

2.1.7 Rayleigh bottom loss 22

2.1.8 Reverberation and noise measurements 24

2.1.9 Sound speed 25

2.1.10 Surface back scatter 28

2.1.11 Surface loss 31

2.1.12 Surface reflection angle 34

2.1.13 Target strength 35

2.1.14 Volume back scatter 37

2.1.15 Wave height 39

2.1.16 Wind speed measurements 40

2.2 Platform 41

2.2.1 Sensor 42

3 Initiate calculation **46**

4 Calculation results **47**

Code examples **55**

A.1 C# code example 55

A.2 Matlab code example 58

A.3 C++ example code using LybinCom 60

Abbreviations **70**

Type definitions **70**

References **71**

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 [6]. LybinCom 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 LybinCom 7.0. 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 Microsoft COM technology enables LybinCom to be used from 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 Matlab.

2 LYBIN model data

The LybinModelData class contains all the model data to be used in a simulation: the environment, the platform and all the parameters controlling the acoustic calculations. All the parameters and functions in LybinModelData are listed in Table 2.1 and Table 2.2.

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 metres and a maximum depth step size of 5 metres. 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 LybinCom use certain datasets instead of predefined default values. In order to follow these demands, the specified datasets must be sent into LybinCom. If LybinCom cannot find these datasets, the switches will be set back to default values.

Table 2.1 Parameters in the LybinModelData class.

Parameter	Type	Default value	Unit
DepthCells *Number of depth cells in the calculation output.* *DepthCell is read only, so the function SetDepthScaleAndDepthCells must be used to set this parameter directly.*	Integer	50	
DepthCellSize *Size of the depth cells in the calculation output.* *DepthCellSize is read only, so the functions SetDepthCellSizeAndDepthSteps or SetDepthScaleAndDepthCellSize must be used to set this parameter directly.*	Double	6	metres

Parameter	Type	Default value	Unit
DepthScale *Maximum depth in the calculation.*	Double	300	metres
DepthSteps *Number of depth steps to be used during the calculation.* *DepthSteps is read only, so the functions SetDepthCellSizeAndDepthSteps or SetDepthScaleAndDepthCellSteps must be used to set this parameter directly.*	Integer	1000	
DepthStepSize *Size of the depth steps to be used during the calculation.* *This parameter is read only, and is derived by other depth calculation parameters.*	Double	0.3	metres
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	metres
ImpulseResponseWindowHeight *The heigth of the window that the impulse response will be calculated from.*	Double	80	metres
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	

Parameter	Type	Default value	Unit
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.* *RangeCells is read only, so the function SetRangeScaleAndRangeCells must be used to set this parameter directly.*	Integer	50	
RangeCellSize *Size of the range cells in the calculation output.* *RangeCellSize is read only, so the functions SetRangeCellSizeAndRangeSteps or SetRangeScaleAndRangeCellSize must be used to set this parameter directly.*	Double	200	metres
RangeScale *Maximum range in the calculation.*	Double	10000	metres
RangeSteps *Number of range steps to be used during the calculation.* *RangeSteps is read only, so the functions SetRangeCellSizeAndRangeSteps or SetRangeScaleAndRangeCellSteps must be used to set this parameter directly.*	Integer	500	
RangeStepSize *Size of the range steps to be used during the calculation.* *RangeStepSize is read only, and it is derived by other range calculation parameters.*	Double	20	metres

Parameter	Type	Default value	Unit
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	degrees
DoTravelTimeCalculation *Switch to control whether to calculate ray 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	
UseMeasuredBeamPattern *Tells the model to use measured beam pattern.*	Boolean	false	

Parameter	Type	Default value	Unit
UseMeasuredBottomLoss *Tells the model 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	
UseRayleighBottomLoss *Tells the model to calculate bottom loss according to the Rayleigh bottom loss algorithms.* *If UseRayleighBottomLoss = true, it will overrule UseMeasuredBottomLoss.*	Boolean	false	

Parameter	Type	Default value	Unit
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 alloved 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 2.2 Functions in the LybinModelData class

Function	Type	Unit of input parameters
ChangeModelData(string xmlData) *Send in the complete XML LYBIN dataset as one string.*	Void	
GetCurrentModelData(out string modelData) *Get the complete XML LYBIN dataset as one string.*	Void	
SetDepthCellSizeAndDepthSteps(double cellSize, int steps) *Set the depth cell size and the number of depth steps. This setting will overrule all earlier depth settings affecting the calculation precision.*	Void	**cellSize**: meters

Function	Type	Unit of input parameters
SetDepthScaleAndDepthCells(double scale, int cells) *Set the depth scale and the number of depth cells. This setting will overrule all earlier depth settings affecting the calculation precision.*	Void	**scale**: meters
SetDepthScaleAndDepthCellSize(double scale, double cellSize) *Set the depth scale and the depth cell size. This setting will overrule all earlier depth settings affecting the calculation precision.*	Void	**scale**: meters, **cellSize**: meters
SetDepthScaleAndDepthSteps(double scale, int steps) *Set the depth scale and the number of depth steps. This setting will overrule all earlier depth settings affecting the calculation precision.*	Void	**scale**: meters
SetRangeCellSizeAndRangeSteps(double cellSize, int steps) *Set the range cell size and the number of range steps. This setting will overrule all earlier range settings affecting the calculation precision.*	Void	**cellSize**: meters
SetRangeScaleAndRangeCells(double scale, int cells) *Set the range scale and the number of range cells. This setting will overrule all earlier range settings affecting the calculation precision.*	Void	**scale**: meters
SetRangeScaleAndRangeCellSize(double scale, double cellSize) *Set the range scale and the range cell size. This setting will overrule all earlier range settings affecting the calculation precision.*	Void	**scale**: meters, **cellSize**: meters
SetRangeScaleAndRangeSteps(double scale, int steps) *Set the range scale and the number of range steps. This setting will overrule all earlier range settings affecting the calculation precision.*	Void	**scale**: meters

2.1 Environment

The environment class contains all the environmental data as listed in Table 2.3.

LybinCom can handle range dependent environments. In LybinCom, 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 is only limited by the given calculation accuracy.

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

Table 2.3 The environment class holds all the environment data.

Parameter	Type
BottomBackScatter	Range dependent bottom back scatter values as a function of each rays grazing angle with the bottom.
BottomLoss	Range dependent bottom loss values as a function of each rays grazing angle with the bottom.
BottomProfile	Single measurement of depth as a function of range.
BottomType	Range dependent bottom types ranging from 0-10. The bottom type is transformed into bottom loss before it is used in model calculations.
LambertsCoefficient	Range dependent bottom back scatter model parameter according to Lamberts law.
Ocean	Parameters describing the media (ocean) and the assumed target, such as ambient noise, pH, surface scatter, target strenght and ship dencity.
RayleighBottomLoss	Rayleigh bottom loss model parameters.

ReverberationAndNoise	Range dependent total reverberation and noise data measurements.
SoundSpeed	Range dependent sound speed, temperature, and salinity measurements as a function of depth.
SurfaceBackScatter	Range dependent surface back scatter values as a function of each rays grazing angle with the sea surface.
SurfaceLoss	Range dependent surface loss values as a function of each rays grazing angle with the sea surface.
SurfaceReflectionAngle	Range dependent surface reflection angle values.
VolumeBackScatter	Range dependent volume back scatter values.
WaveHeight	Range dependent wave height measurements.
WindSpeed	Range dependent wind speed measurements.

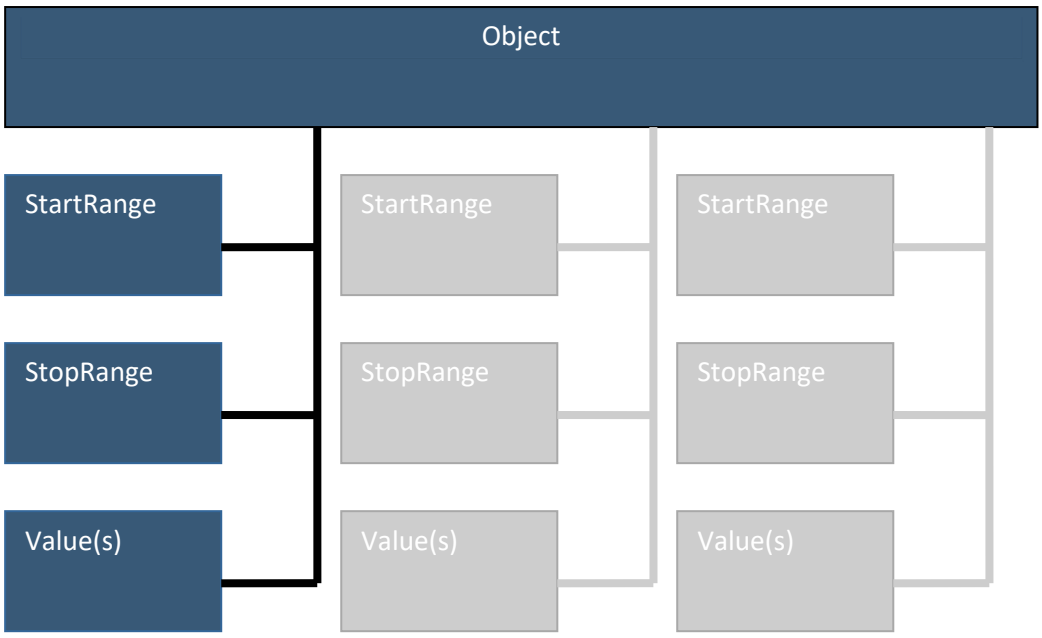

Figure 2.1 Schematic description of a range dependent object with start and stop parameters.

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 LybinCom will use interpolation to produce data for intermediate ranges points. The start and stop functionality might be utilized to illustrate meteorological or oceanographic fronts, entering ranges 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.

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 LybinCom as single values with corresponding range. The number of data points in each dataset is optional.

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 LybinCom in tabular form, giving backscattering coefficients (in dB) for a set of grazing angles. Based on the tabulated values, LybinCom 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. LybinCom will only use the bottom back scatter values given if the TypeOfRevNoiseCalculation parameter in LybinModelData class is set to 1 (Calculate bottom reverberation from back scatter values).

There is only one parameter in the BottomBackScatter class, the BottomBackScatterTableCount, given in Table 2.4. The functions in BottomBackScatter class are given in Table 2.5.

An example of how the some of the bottom back scatter functions can be used is shown in Figure 2.2. In the example, two bottom back scatter tables are set at different ranges. At the end of the example, the first bottom back scatter table is fetched back from LybinCom.

Table 2.4 Parameters in the BottomBackScatter class.

Parameter	Type	Default value	Unit
BottomBackScatterTableCount *Number of bottom back scatter tables.*	Integer	1	

Table 2.5 Functions in the BottomBackScatter class.

Function	Type	Unit of input parameters
AddBottomBackScatterTable(int start, int stop, object table) *Add another bottom back scatter table. This function can only be used once the first bottom back scatter table is added with the SetFirstBottomBackScatterTable function.*	Void	**start**: metres, **stop**: metres **table**: dB/meter^2 vs. degrees
GetBottomBackScatterTable(int index, out int start, out int stop, out object table) *Get the bottom back scatter table corresponding to the given index.*	Void	
SetFirstBottomBackScatterTable(int start, int stop, object table) *Set the first bottom back scatter table.*	Void	

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 LybinCom in tabular form, giving bottom loss (in dB) for a set of grazing angles. Based on the tabulated values, LybinCom interpolates between tabulated values to create loss values for grazing angles in between the given angles.

The parameter UseMeasuredBottomLoss tells LybinCom 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 LybinCom. Both these parameters can be found in the LybinModelData class.

There is only one parameter in the BottomLoss class, the BottomLossTableCount, given in Table 2.6. The functions in the BottomLoss class are given in Table 2.7.

```
// The first bottom back scatter table
var bb = new double[3, 2];
bb[0, 0] = 11;
bb[0, 1] = 3.2;
bb[1, 0] = 33;
bb[1, 1] = 7.4;
bb[2, 0] = 88;
bb[2, 1] = 4;

// The next bottom back scatter table
var bbn = new double[3, 2];
bbn[0, 0] = 10;
bbn[0, 1] = 6.2;
bbn[1, 0] = 43;
bbn[1, 1] = 7.2;
bbn[2, 0] = 77;
bbn[2, 1] = 6.8;

_lybin.SetFirstBottomBackScatterTable(0,300, bb);
_lybin.AddBottomBackScatterTable(400, 4000, bbn);;
_lybin.TypeOfRevNoiseCalculation = 1;

// Get the first bottom back scatter table
const int index = 0;
_lybin.GetBottomBackScatterTable(
      index, out var start, out var stop, out var table);
```

Figure 2.2 C# code example: Two bottom back scatter tables are added to _lybin with their range of validity. Then the first table containing range dependent bottom back scatter values is fetched back from _lybin.

Table 2.6 Parameters in the BottomLoss class.

Parameter	Type	Default value	Unit
BottomLossTableCount *Number of bottom loss tables.*	Integer	1	

Table 2.7 Functions in the BottomLoss class.

Function	Type	Unit of input parameters
AddBottomLossTable(int start, int stop, object table) *Add another bottom loss table.This function can only be used once the first bottom loss table is added with the SetFirstBottomLossTable function.*	Void	**start**: metres, **stop**: metres **table**: dB vs. degrees
GetBottomLossTable(int index, out int start, out int stop, out object table) *Get the bottom loss table corresponding to the given index.*	Void	
SetFirstBottomLossTable(int start, int stop, object table) *Set the first bottom loss table.*	Void	

An example of how some of the bottom loss functions can be used is shown below in Figure 2.3. In the example, the first bottom loss table is set to be valid from 0 km to 3 km. The loss table consist of the following data: 10° : 4.2 dB, 30° : 6.4 dB and 56° : 9 dB. At the end of the example, the first bottom loss table is fetched back from LybinCom.

```
// Set the first bottom loss fan
double[,] bl = new double[3, 2];
bl[0, 0] = 10;
bl[0, 1] = 4.2;
bl[1, 0] = 30;
bl[1, 1] = 6.4;
bl[2, 0] = 56;
bl[2, 1] = 9;

_lybin.SetFirstBottomLossTable(0, 3000, bl);
_lybin.UseMeasuredBottomLoss = true;
_lybin.UseRayleighBottomLoss = false;

// Get the first bottom loss table
const int index = 0;
_lybin.GetBottomLossTable(
    index, out var start, out var stop, out var table);
```

Figure 2.3 C# code example: A bottom loss table is added to _lybin with its range of validity. Then the first table containing range dependent bottom loss values is fetched back from _lybin.

2.1.3 Bottom profile

The BottomProfile class only has one accessible parameter, the BottomProfile, which is listed in Table 2.8. The BottomProfile can consist of any number points in range with their corresponding bottom depths.

Table 2.8 Parameter in the bottom profile class.

Parameter	Type	Default value	Unit
BottomProfile *Depth of bottom as function of range.*	Object (Double[x,2])	(0, 280) (*range, depth*)	(metres, metres)

An example on how the BottomProfile can be used is shown in Figure 2.4. In the example, two points are inserted. The first is the depth 300 metres at a range of 0 meter. The second is the depth 380 metres at a range of 1000 metres.

```
double[,] bp = new double[2, 2];
bp[0, 0] = 0;
bp[0, 1] = 300;
bp[1, 0] = 1000;
bp[1, 1] = 380;
_lybin.BottomProfile = bp;
```

Figure 2.4 C# code example: Bottom depth values are added to the BottomProfile.

2.1.4 Bottom type

The geo-acoustic properties of the bottom are coded by a single parameter in LybinCom. 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. LybinCom will use the given bottom type when the TypeOfRevNoiseCalculation parameter in LybinModelData class is set to 0: (Calculate bottom reverberation from bottom types).

The BottomType class only has one accessible parameter, BottomType, which is listed in Table 2.9.

Table 2.9 Parameters in the BottomType class.

Parameter	Type	Default value	Unit
BottomType	Object (Double[x,3])	(0, 0, 0) (*start, stop, value)*	(metres, metres, -)

An example of how BottomType can be used is shown in Figure 2.5. In the example two different bottom types are set. From the range of 0 km to 5 km, the bottom type is 4. From the range of 5 km to 10 km, the bottom type is 2.3.

```
// Set the bottom type
double[,] bt = new double[2, 3];
bt[0, 0] = 0;
bt[0, 1] = 5000;
bt[0, 2] = 4;
bt[1, 0] = 5000;
bt[1, 1] = 10000;
bt[1, 2] = 2.3;

_lybin.BottomType = bt;
```

Figure 2.5 C# code example: Bottom type values are added with their range of validity.

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 supplemental start and stop values. If LambertsCoefficient is to be used, the parameter TypeOfRevNoiseCalculation has to be set to 3: (Calculate bottom reverberation using Lamberts rule). The parameter TypeOfRevNoiseCalculation can be found in the LybinModelData class.

The LambertsCoefficient class has only one accessible parameter, LambertsCoefficient, which is listed in Table 2.10. The range dependent Lamberts coefficient can be added to the LambertCoefficient class as seen in the C# example in Figure 2.6. The default LambertsCoefficient value is 0 dB.

Table 2.10 Parameters in the LambertsCoefficient class.

Parameter	Type	Default values	Units
LambertsCoefficient *Lamberts coefficient to be used in calculation og bottom back scattering values.*	Object *(Double[x,3])*	(0, 0, 0) (*start, stop, value)*	(metres, metres, dB)

```
// Lamberts rule
double[,] lc = new double[2, 3];
lc[0, 0] = 0;
lc[0, 1] = 5000;
lc[0, 2] = -20;
lc[1, 0] = 5000;
lc[1, 1] = 10000;
lc[1, 2] = -27;

_lybin.LambertsCoefficient = lc;
_lybin.TypeOfRevNoiseCalculation = 3;
```

Figure 2.6 C# code example: Lamberts coefficient values are added with their range of validity.

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 2.11.

Both Ambient noise and target strength can either be given as fixed parameters or they 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.

Table 2.11 Parameters in the Ocean class.

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* *The target is within the main lobe of the sonar, and the targets Doppler speed is lower than the ships own speed.*	Enum	1	

Parameter	Type	Default value	Unit
1: Typical *Typical horizontal side lobe suppression, where the targets Doppler speed is lower than the ships own speed.* *2: NoReverb* *The Doppler speed of the target is equal or higher than the ships own speed, so the scenario become noise limited.*			
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 *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.*	Boolean	true	
TargetAspectAngle *Aspect angle of target.*	Double	0	degrees
TargetCource *Cource of target.*	Double	0	degrees
TargetSpeed *Speed of target.*	Double	10	metres/ second
TargetStrength *Target echo strenght.*	Double	10	dB

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 parameter 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 2.12. A C# example where Rayleigh bottom loss values are added to the RayleighBottomLoss class is shown in Figure 2.7.

In order to make LybinCom 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.

Table 2.12 Parameters in the RayleighBottomLoss class.

Parameter	Type	Default value	Unit
RayleighBottomLoss	Object (Double[1,3])	(0.5, 1700, 2.0) (*bottom attenuation, bottom sound speed, density ratio between density in the bottom and density in water)*	dB/wavelength, metres/second, scalar

```
double bottomAttenuation = 0.8;
double bottomSoundSpeed = 1706;
double densityRatio = 4;

double[,] rbl = new double[1, 3];
rbl[0, 0] = bottomAttenuation;
rbl[0, 1] = bottomSoundSpeed;
rbl[0, 2] = densityRatio;

_lybin.RayleighBottomLoss = rbl;
_lybin.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, LybinCom uses linear interpolation.

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

The ReverberationAndNoiseSample can consist of any reverberation and noise samples containing range and depth values as listed in Table 2.13. The samples can be added to the ReverberationAndNoise class as seen in the C# example in Figure 2.8.

Table 2.13 Parameter in the ReverberationAndNoiseMeasurements class.

Parameter	Type	Default value	Unit
ReverberationAndNoiseMeasurements *Reverberation and noise measurement as function of range.*	Object (Double[x,2])	(0, 80) (*range, value*)	(metres, dB)

```
double[,] rn = new double[2, 2];
rn[0, 0] = 0;
rn[0, 1] = 70;
rn[1, 0] = 5000;
rn[1, 1] = 50;

_lybin.ReverberationAndNoiseMeasurments = rn;
_lybin.TypeOfRevNoiseCalculation = 2;
```

Figure 2.8 C# code example: Reverberation and noise values are added to the ReverberationAndNoiseMeasurements class.

2.1.9 Sound speed

The SoundSpeed class handles the sound speed in the water volume. The sound speed is a function of both range and depth. Since the sound speed is most often measured as depth dependant profiles, the SoundSpeed class can contain multiple sound speed profiles, representative of different ranges.

The profile can contain the parameters temperature, salinity and sound speed for a given set of depths. If two of the three parameters are given, LybinCom will estimate the remaining one based on depth and the two given parameters. If only one parameter is available, LybinCom can estimate the missing parameters using depth and a default value. Sound speed, temperature and salinity have default values. They are listed in Table 2.14. If only temperature is given, the default salinity is used to calculate the sound speed. If only sound speed is given, the default salinity is used to calculate temperature. If only salinity is given the default sound speed is used to calculate the temperature. Sound speeds for intermediate depths are computed using linear interpolation.

Table 2.14 Default values for profile parameters in the SoundSpeed class.

Parameter	Default value	Unit
SoundSpeed	1480	metres/second
Temperature	7,36	°Celsius
Salinity	35	parts per thousand

There is only one parameter in the SoundSpeed class, the SoundSpeedProfileCount, given in Table 2.15. The functions in the SoundSpeed class are given in Table 2.16. Depth is always the first parameter in a profile. The internal order of the others is given in the function name, and is:

1. Sound speed
2. Temperature
3. Salinity

Table 2.15 Parameters in the SoundSpeed class.

Parameter	Type	Default value	Unit
SoundSpeedProfileCount *Number of sound speed profiles.*	Integer	1	

Table 2.16 Functions in the SoundSpeed class.

Function	Type	Unit of input parameters
AddSalinityProfile(int start, int stop, object profile) *Add another salinity profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	**start**: metres **stop**: metres **profile:** **depth**: metres **sound speed**: metres/second **temperature**: °Celsius **salinity**: parts per thousand (ppt)
AddSoundSpeedProfile(int start, int stop, object profile) *Add another sound speed profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	
AddSoundSpeedAndSalinityProfile(int start, int stop, object profile) *Add another sound speed and salinity profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	
AddSoundSpeedAndTempProfile(int start, int stop, object profile) *Add another sound speed and temperature profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	
AddSoundSpeedTempAndSalinityProfile(int start, int stop, object profile) *Add another sound speed, temperature and salinity profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	
AddTempAndSalinityProfile(int start, int stop, object profile) *Add another temperature and salinity profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	

Function	Type	Unit of input parameters
AddTempProfile(int start, int stop, object profile) *Add another temperature profile. This function can only be used after the first profile has been added with one of the SetFirstProfile functions.*	Void	
GetSoundSpeedProfile(int index, out int start, out int stop, out object profile) *Get the sound speed profile corresponding to the given index.*	Void	
SetFirstSalinityProfile(int start, int stop, object profile) *Set the first salinity profile.*	Void	
SetFirstSoundSpeedProfile(int start, int stop, object profile) *Set the first sound speed profile.*	Void	
SetFirstSoundSpeedAndSalinityProfile(int start, int stop, object profile) *Set the first sound speed and salinity profile.*	Void	
SetFirstSoundSpeedAndTempProfile(int start, int stop, object profile) *Set the first sound speed and temperature profile.*	Void	
SetFirstSoundSpeedTempAndSalinityProfile (int start, int stop, object profile) *Set the first sound speed, temperature and salinity profile.*	Void	
SetFirstTempAndSalinityProfile(int start, int stop, object profile) *Set the first temperature and salinity profile.*	Void	
SetFirstTempProfile(int start, int stop, object profile) *Set the first temperature profile.*	Void	

An example of how some of the sound speed functions can be used is shown in Figure 2.9. In the example, the first sound speed profile is set at the range from 0 to 2 kilometres, LybinCom is to use the profile given by the sound speed 1480 m/s, temperature 7° Celsius and a salinity of 35 ppt at 0 metres depth and the sound speed 1510 m/s, temperature 8° Celsius and a salinity of 34 ppt at 620 metres depth.

The second sound speed profile is to be used at ranges from 2 km to 5 km. This profile contains only sound speed measurements. At the depth of 50 m, the sound speed is 1488 m/s, and at the depth of 100 m the sound speed is 1499 m/s.

The third profile contains temperature and salinity measurements and is to be used at the ranges from 5 km to 8 km. At the depth of 10 m, the temperature is 6.1° Celsius and the salinity is 34 ppt. At the depth of 200 m, the temperature is 4.2° Celsius and the salinity is 33 ppt. At the end of the example, the first sound speed profile is retrieved from LybinCom. This profile contains calculated temperature, salinity and sound speed as used in the calculations.

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 LybinCom, giving backscattering coefficients (in dB) for the incoming rays hitting the sea surface. Based on the values, LybinCom 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. LybinCom will only use the surface back scatter values given if the UseMeasuredSurfaceBackScatter parameter in LybinModelData class is set to true.

There is only one parameter in the SurfaceBackScatter class, the SurfaceBackScatterTableCount, given in Table 2.17. The functions in SurfaceBackScatter class are given in Table 2.18.

An example of how the some of the surface back scatter functions can be used is shown in Figure 2.10. In the example, two surface back scatter tables are set at different ranges.

```
// Set the first sound speed profile
// Containing sound speed, temperature and salinity
double[,] ssp = new double[2, 4];
ssp[0, 0] = 0;    // Depth
ssp[0, 1] = 1480; // Sound speed
ssp[0, 2] = 7;    // Temperature
ssp[0, 3] = 35;   // Salinity
ssp[1, 0] = 620;  // Depth
ssp[1, 1] = 1510; // Sound speed
ssp[1, 2] = 8;    // Temperature
ssp[1, 3] = 34;   // Salinity
_lybin.SetFirstSoundSpeedTempAndSalinityProfile(0, 2000, ssp);

// Set the second sound speed profile
// Containing only sound speed
double[,] sss = new double[2, 2];
sss[0, 0] = 50;   // Depth
sss[0, 1] = 1488; // Sound speed
sss[1, 0] = 100;  // Depth
sss[1, 1] = 1499; // Sound speed
_lybin.AddSoundSpeedProfile(2000, 5000, sss);

// Set the third sound speed profile
// Containing temperature and salinity
double[,] tsp = new double[2, 3];
tsp[0, 0] = 10;   // Depth
tsp[0, 1] = 6.1;  // Temperature
tsp[0, 2] = 34;   // Salinity
tsp[1, 0] = 200;  // Depth
tsp[1, 1] = 4.2;  // Temperature
tsp[1, 2] = 33;   // Salinity
_lybin.AddTempAndSalinityProfile(5000, 8000, tsp);

// Get the first SoundSpeedProfile
int index = 0;
int start, stop;
object profile = new object();
_lybin.GetSoundSpeedProfile(
      index, out start, out stop, out profile);
```

Figure 2.9 C# code example: Sound speed values are added to the SoundSpeedProfile.

Table 2.17 Parameters in the SurfaceBackScatter class.

Parameter	Type	Default value	Unit
SurfaceBackScatterTableCount *Number of surface back scatter tables.*	Integer	1	

Table 2.18 Functions in the SurfaceBackScatter class.

Function	Type	Unit of input parameters
AddSurfaceBackScatterTable(int start, int stop, object table) *Add another surface back scatter table. This function can only be used once the first surface back scatter table is added with the SetFirstSurfaceBackScatterTable function.*	Void	**start**: metres, **stop**: metres **table**: dB/meter^2 vs. degrees
GetSurfaceBackScatterTable(int index, out int start, out int stop, out object table) *Get the surface back scatter table corresponding to the given index.*	Void	
SetFirstSurfaceBackScatterTable(int start, int stop, object table) *Set the first surface back scatter table.*	Void	

```
// The first surface back scatter table
var sbs = new double[3, 2];
sbs[0, 0] = 11;
sbs[0, 1] = 3.2;
sbs[1, 0] = 33;
sbs[1, 1] = 7.4;
sbs[2, 0] = 88;
sbs[2, 1] = 4;

// The next surface back scatter table
var sbb = new double[3, 2];
sbb[0, 0] = 10;
sbb[0, 1] = 6.2;
sbb[1, 0] = 43;
sbb[1, 1] = 7.2;
sbb[2, 0] = 77;
sbb[2, 1] = 6.8;

_lybin.SetFirstSurfaceBackScatterTable(0, 300, sbs);
_lybin.AddSurfaceBackScatterTable(400, 4000, sbb);
_lybin.UseMeasuredSurfaceBackScatter = true;
```

Figure 2.10 *C# code example: Two surface back scatter tables are added to _lybin with their range of validity.*

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. A dataset representing surface loss is entered into LybinCom, giving surface loss in dB for a set of grazing angles. Based on the values, LybinCom interpolates to create loss values for all grazing angles.

The parameter UseMeasuredSurfaceLoss tells LybinCom 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 LybinCom. UseMeasuredSurfaceLoss can be found in the LybinModelData class.

There is only one parameter in the SurfaceLoss class, the SurfaceLossTableCount, given in Table 2.19. The functions in the SurfaceLoss class are given in Table 2.20.

Table 2.19 Parameters in the SurfaceLoss class.

Parameter	Type	Default value	Unit
SurfaceLossTableCount *Number of surface loss tables.*	Integer	1	

Table 2.20 Functions in the SurfaceLoss class

Function	Type	Unit of input parameters
AddSurfaceLossTable(int start, int stop, object table) *Add another surface loss table. This function can only be used once the first surface loss table is added with the SetFirstSurfaceLossTable function.*	Void	**start**: metres, **stop**: metres **table**: dB vs. degrees
GetSurfaceLossTable(int index, out int start, out int stop, out object table) *Get the surface loss table corresponding to the given index.*	Void	
SetFirstSurfaceLossTable(int start, int stop, object table) *Set the first surface loss table.*	Void	

An example of how some of the surface loss functions can be used is shown below in Figure 2.11. In the example, the first surface loss fan is set to be valid from 0 km to 3 km. The loss table consist of the following data: 10°= 4.2 dB, 30° = 6.4 dB and 56° = 9 dB. At the end of the example, the first surface loss table is fetched back from LybinCom.

```
// Set the first surface loss fan
double[,] sl = new double[3, 2];
sl[0, 0] = 10;
sl[0, 1] = 4.2;
sl[1, 0] = 30;
sl[1, 1] = 6.4;
sl[2, 0] = 56;
sl[2, 1] = 9;

_lybin.SetFirstSurfaceLossTable(0, 30, sl);
_lybin.UseMeasuredSurfaceLoss = true;

// Get the first surface loss table
index = 0;
_lybin.GetSurfaceLossTable(
      index, out var start, out var stop, out var table);
```

Figure 2.11 C# code example: A surface loss table is added to _lybin with its range of validity. Then the first table containing range dependent surface loss values is fetched back from _lybin.

2.1.12 Surface reflection angle

Predefined surface reflection angles can be set as seen in the C# example in Figure 2.12. The SurfaceReflectionAngle class only has one accessible parameter, BottomType, which is listed in Table 2.21.

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.

Table 2.21 Parameters in the SurfaceReflectionAngle class.

Parameter	Type	Default value	Unit
SurfaceReflectionAngle	Object (Double[x,3])	(0, 0, 0) (*start, stop, value)*	(metres, metres, degrees)

```
// Set the surface reflection angle
double[,] sra = new double[2, 3];
sra[0, 0] = 0;
sra[0, 1] = 5000;
sra[0, 2] = 30;
sra[1, 0] = 6000;
sra[1, 1] = 8000;
sra[1, 2] = 40;

_lybin.SurfaceReflectionAngle = sra;
_lybin.UseMeasuredSurfaceReflectionAngles = true;
```

Figure 2.12 C# code example: range dependent surface reflection angles are 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 LybinCom 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.

There is only one parameter in the TargetStrength class, the TargetStrengthTableCount, given in Table 2.22. The functions in the TargetStrength class are given in Table 2.23. An example of how some of the target strength functions can be used is shown below in Figure 2.13.

Table 2.22 Parameters in the TargetStrength class.

Parameter	Type	Default value	Unit
TargetStrengthTableCount *Number of target strangth tables.*	Integer	1	

Table 2.23 Functions in the TargetStrength class

Function	Type	Unit of input parameters
AddTargetStrengthTable(int start, int stop, object table) *Add another target strength table. This function can only be used once the first target strength table is added with the SetFirstTargetStrengthTable function.*	Void	**start**: Hz, **stop**: Hz **table**:
GetTargetStrengthTable(int index, out int start, out int stop, out object table) *Get the target strength table corresponding to the given index.*	Void	

SetFirstTargetStrengthTable(int start, int stop, object table) *Set the first target strength table.*	Void	dB vs. degrees

```
// Set the first target strength table
var ts = new double[3, 2];
ts[0, 0] = 11;
ts[0, 1] = 3.2;
ts[1, 0] = 33;
ts[1, 1] = 7.4;
ts[2, 0] = 88;
ts[2, 1] = 4;
_lybin.SetFirstTargetStrengthTable(0, 3000, ts);

// Add a second target strength table
var ts2 = new double[3, 2];
ts2[0, 0] = 1;
ts2[0, 1] = 3.0;
ts2[1, 0] = 199;
ts2[1, 1] = 7.4;
ts2[2, 0] = 200;
ts2[2, 1] = 42;
_lybin.AddTargetStrengthTable(4000, 7000, ts2);

_lybin.UseMeasuredTargetStrength = true;
_lybin.TargetAspectAngle = 300;

// Get the second target strength table
var index = 1;
_lybin.GetTargetStrengthTable(
      index, out start, out stop, out table);
```

Figure 2.13 C# code example: range dependent 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 LybinCom, 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.

There is only one parameter in the VolumeBackScatter class, the VolBackScatterProfileCount, given in Table 2.24. The functions in the VolumeBackScatter class are given in Table 2.25.

Table 2.24 Parameters in the VolumeBackScatter class.

Parameter	Type	Default value	Unit
VolBackScatterProfileCount *Number of volume back scatter profiles.*	Integer	1	

Table 2.25 Functions in the VolumeBackScatter class.

Function	Type	Unit of input parameters
AddVolBackScatterProfile(int start, int stop, object profile) *Add another volume back scatter profile.This function can only be used when the first volume back scatter profile is added with the SetFirstVolumeBackScatterProfile function.*	Void	**start**: meters, **stop**: meters **profile**: dB /meter^3
GetVolBackScatterProfile(int index, out int start, out int stop, out object profile) *Get the volume back scatter profile corresponding to the given index.*	Void	
SetFirstVolBackScatterProfile(int start, int stop, object profile) *Set the first volume back scatter profile.*	Void	

An example of how some of the volume back scatter functions can be used is shown in Figure 2.14. In the example, the first volume back scatter profile is set. At the range from 0 km to 1 km, LybinCom is to use the values: 10 meters = -80 dB and 50 meters = -92 dB. At the end of the example, the first volume back scatter profile is fetched back from LybinCom. Volume reverberation back scatter coefficients are given as dB per cubic metre.

```
// Set the first volume back scatter profile
double[,] vc = new double[2, 2];
vc[0, 0] = 10;
vc[0, 1] = -80;
vc[1, 0] = 50;
vc[1, 1] = -92;
_lybin.SetFirstVolBackScatterProfile(0, 1000, vc);

// Get the first volume back scatter profile
int index = 0;
int start, stop;
object profile = new object();
_lybin.GetVolBackScatterProfile(
      index, out start, out stop, out profile);
```

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

2.1.15 Wave height

The WaveHeight class only has one accessible parameter, the WaveHeight, which is listed in Table 2.26.

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.

Table 2.26 Parameters in the WaveHeight class.

Parameter	Type	Default value	Unit
WaveHeight *Wave height in the area of calculation.*	Object (Double[x,3])	(0, 0, 0) (*start, stop, value)*	(metres, metres, metres)

An example of how WaveHeight can be used is shown Figure 2.15. In the example the wave height is 5 metres from 0 to 4 kilometres, and 3 metres from 4 to 9 kilometres.

```
// Set the wave height
 double[,] wh = new double[2, 3];
 wh[0, 0] = 0;
 wh[0, 1] = 4000;
 wh[0, 2] = 5;
 wh[1, 0] = 4000;
 wh[1, 1] = 9000;
 wh[1, 2] = 3;

 _lybin.WaveHeight = wh;
 _lybin.UseWaveHeight = false;
```

Figure 2.15 C# code example: range dependent wave height values are added to the WaveHeight class.

2.1.16 Wind speed measurements

The WindspeedMeasurement class only has one accessible parameter, WindSpeedMeasurements, which is listed in Table 2.27.

Table 2.27 Parameters in the WindSpeedMeasurement class.

Parameter	Type	Default values	Units
WindSpeedMeasurments *Wind speed in the area of calculation.*	Object *(Double[x,3])*	(0, 0, 0) (*start, stop, value)*	(metres, metres, metres/Second)

An example of how WindSpeedMeasurements can be used is shown in the C# code example in Figure 2.16. In the example, the measured wind speed is 10 meters/second from 0 to 5 kilometres, and 6 meters/second from 5 to 10 kilometres.

```
// Set the wind speed measurement
double[,] ws = new double[2, 3];
ws[0, 0] = 0;
ws[0, 1] = 5000;
ws[0, 2] = 10;
ws[1, 1] = 5000;
ws[1, 1] = 10000;
ws[1, 2] = 6;

_lybin.WindSpeedMeasurments = ws;
```

Figure 2.16 C# code example: range dependent wind speed measurements are added to the WindSpeedMeasurements 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 2.28.

Parameter	Type	Default value	Unit
Latitude *Actual latitude of platform.*	Double	0	degrees north
ShipCourse *Platform course relative to north.*	Double	0	degrees
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 2.28 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 2.29.

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		
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	metres

Parameter	Type	Default value	Unit
DetectionThreshold *The strength of the signal relative to the masking level necessary to see an object with the sonar.*	Double	10	dB
DirectivityIndex *The sonar's 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
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	

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

Table 2.29 Parameters in the sensor class.

2.2.1.1 BeamPattern

The BeamPattern measurement is an optional choice where one uses values instead of letting LybinCom estimate 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, LybinCom 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 parameters in the BeamPattern class are given in Table 2.30.

Table 2.30 Parameters in the BeamPattern class.

Parameter	Type	Default value	Unit
BeamPatternReceiver *BeamPattern of the receiver*	Object (Double[x,2])	(0, 0) (*angle, value)*	(degrees, dB)

BeamPatternSender *BeamPattern of the sender*	Object (Double[x,2])	(0, 0) (*angle, value)*	(degrees, dB)

```
// Add beam patterns for sender and receiver
var bps = new double[3, 2];
bps[0, 0] = 10;
bps[0, 1] = 10;
bps[1, 0] = 30;
bps[1, 1] = 30;
bps[2, 0] = 80;
bps[2, 1] = 80;

var bpr = new double[3, 2];
bpr[0, 0] = 10;
bpr[0, 1] = 11;
bpr[1, 0] = 30;
bpr[1, 1] = 33;
bpr[2, 0] = 88;
bpr[2, 1] = 88;

_lybin.BeamPatternReceiver = bpr;
_lybin.BeamPatternSender = bps;
_lybin.UseMeasuredBeamPattern = true;
```

Figure 2.17 C# code example: two BeamPattern samples 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 2.31* 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
Form *Pulse type:* *FM: Frequency modulated* *CW: Continuous wave*	String	FM	
Length *Pulse length.*	Double	60	Milliseconds

Table 2.31 Parameters in the pulse class.

3 Initiate calculation

The DoCalculation function initiates a new LYBIN calculation. Before the DoCalculation function is called, all input parameters must be set, otherwise default parameters are used.

If a calculation fails, DoCalculation is implemented to throw an exception containing a message describing the cause of the error.

Table 3.1 Function for initiation of calculation.

Function	Type
DoCalculation() *Start the calculation.*	Void

4 Calculation results

The calculation results can be accessed through parameters or functions found in LybinModelData. The result parameters are listed in Table 4.1. Each parameter represents a complete dataset. The result functions give more flexibility in the way that you can access the calculated results. If a calculation fails, the returned value properties will be NULL.

All the functions delivering calculation results are listed in Table 4.2. Available values of the different result categories, resultCat, are listed in Table 4.3 with description.

Table 4.1 Parameters containing calculation results.

Parameter	Access	Type	Unit
AmbientNoiseLevelUsed *The ambient noise used in the calculations.*	Read	Double	dB
BottomReverberation *Calculated bottom reverberation values.*	Read	Double[*RangeCells*]	dB
EchoLevel *Not yet implemented inside LybinCom. This object will not have any data.*	Read	Double[0,0]	dB
ImpulseResponseNumRanges *Returns total number of equidistant ranges the impulse response is calculated for.*	Read	Integer	
MaskingLevel *Calculated masking level (total reverberation + noise after processing).*	Read	Double[*RangeCells*]	dB
NoiseAfterProcessing *Calculated noise after processing.*	Read	Double	dB
ProbabilityOfDetection *Calculated probability of detection.*	Read	Double[*DepthCells*, *RangeCells*]	%

Parameter	Access	Type	Unit
RayTrace *Not implemented inside LybinCom. This object will not have any data.*	Read	Double[0,0]	
ResultModelData *The model data used during the calculation.*	Read	String	
SignalExcess *Calculated signal excess.*	Read	Double[*DepthCells*, *RangeCells*]	dB
SurfaceReverberation *Calculated surface reverberation.*	Read	Double[*RangeCells*]	dB
TotalReverberation *Calculated total reverberation.*	Read	Double[*RangeCells*]	dB
TransmissionLossReceiver *Calculated transmission loss from the target to the receiver.*	Read	Double[*DepthCells*, *RangeCells*]	dB
TransmissionLossTransmitter *Calculated transmission loss from the transmitter to the target.*	Read	Double[*DepthCells*, *RangeCells*]	dB
TravelTimePathCount *Returns total number of travel time paths calculated.*	Read	Integer	
VisualRayTraceCount *Returns the total number of visual ray trace paths calculated.*	Read	Integer	
VolumeReverberation *Calculated volume reverberation.*	Read	Double[*RangeCells*]	dB

Table 4.2 Functions delivering calculation results.

Function	Result format
GetAllResults(out string xmlResult) *Gets all results from the calculation in a single XML-string. The ray trace, travel time and impulse response are not accessible as XML-strings, so they will not be returned through this function call.*	String
GetImpulseResponseFamilliesAsArray(int pIndex) *Returns all the ray families [1] in the range corresponding to pIndex as an array. Each family has the following order of parameters:* *[x,0] Ray family identifier (string)* *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* *[x,1] Intensity loss (double)* *[x,2] Mean arrival time – first arrival time in seconds (double)* *[x,3] Arrival time standard deviation in seconds (double)* *[x,4] Phase identifier [2] (double)* *[x,5] First arrival in seconds (double)*	Object[x,6]
GetImpulseResponseFamily(int pIndex, int pFamilyIndex, out string pFamilyName, out double pIntensity, out double pMeanArrivalTime, out double pStandardDeviation, out double pPhase, out double pFirstArrival) *Returns the calculated ray family identifier, intensity, mean arrival time, arrival time standard deviation, phase and first arrival from one single ray family.* *pIndex represents the corresponding range.* *pFamilyIndex is the running number of the family at the specified range, resulting from the ray tracing calculation. There is no direct*	String, Double, Double, Double, Double, Double

[1] A ray family is a set of rays that share a unique ray history, a sequence of the following: surface reflection, bottom reflection, upper turning point or lower turning point.
[2] The phase identifier is incremented by 2 each time the ray hits the sea surface. Phase shifts originated from bottom hits or caustics are not accounted for in this release.

Function	Result format
connection between pFamilyIndex and pFamilyName. pFamilyName is the ray family identifier. *pIntensity is the intensity loss.* *pMeanArrivalTime is the mean arrival time – first arrival time in seconds.* *pStandardDeviation is the arrival standard deviation in seconds.* *pPhase is the phase identifier[2].* *pFirstArrival is the time of the first arrival in seconds.*	
GetImpulseResponseNumFamilies(int pIndex) *Returns the number of different ray families[1] at the range corresponding to pIndex.*	Integer
GetImpulseResponseRayStatisticsis(int pIndex) *Returns all the ray families[1] in the range corresponding to pIndex as an array. Each family has the following order of parameters:* *[x,0] Ray family identifier (string)* *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* *[x,1] Intensity loss (double)* *[x,2] Mean arrival time – first arrival time in seconds (double)* *[x,3] Arrival time standard deviation in seconds (double)* *[x,4] Phase identifier[2] (double)* *[x,5] First arrival in seconds (double)* *[x,6] Last arrival in seconds (double)* *[x,7] Mean initial angle in degrees (double)* *[x,8] Standard deviation of initial angle in degrees (double)* *[x,9] Minimum initial angle in degrees (double)* *[x,10] Maximum initial angle in degrees (double)*	Object[x,11]

Function	Result format
GetInterpolatedBottomProfile(out object pProfile) *Get the interpolated bottom profile.*	Double[RangeSteps,2]
GetInterpolatedSoundSpeed(out object pProfile) *Get the smoothed and interpolated sound speed matrix.*	Double[RangeSteps, DepthSteps]
GetResultModelData(out string xmlData) *Gets all the model data used in the calculation in a single XML-string.*	String
GetResults(int resultCat, out string xmlResult) *Gets the result specified in resultCat as a XML-string. The possible choices of resultCat are listed in* Table 4.3.	String
GetResultsBin(int resultCat, out object result) *Gets the result specified in resultCat as an object. The possible choices of resultCat are listed in* Table 4.3.	Format depends on type of returned object. See Table 4.3.
GetResultsBinValue(int resultCat, int xVal, int yVal, out double result) *Get a single value from the result specified in resultCat and by the indexes x and y. The possible choices of resultCat are listed in* Table 4.3.	Double
GetTravelTimePath(int pIndex) *Returns all the points in a travel time path. pIndex is path number.* *Each point in the travel time path contains depth in meters, initial ray angle in degrees, range in meters and travel time in seconds.*	Array of TravelTimePoint
GetTravelTimePathAsDoubleArray(int pIndex) *Returns all the points in a travel path as a double array. pIndex is the path number. Each point has the following order of parameters:* *[x ,0] Initial ray angle (degrees)* *[x ,1] Range (meters)* *[x ,2] Depth (meters)* *[x ,3] Travel time (seconds)*	Double[x,4]

Function	Result format
GetTravelTimePathLength(int pIndex) *Returns the length of a travel time path. pIndex is path number.*	Integer
GetTravelTimePoint(int pIndex, int pPointNum) *Returns the calculated parameters in one single point. pIndex is path number and pPointNum is point number in the path. The travel time point contains depth in meters, initial ray angle in degrees, range in meters and travel time in seconds.*	TravelTimePoint
GetTravelTimePoint2(int pIndex, int pPointNum, out double pInitialAngle, out double pRange, out double pDepth, out double pTraveTime) *Returns the calculated parameters in one single point as parameters. pIndex is path number and pPointNum is point number in the path. pInitialAngle is in degrees, pRange in meters, pDepth in meters and pTravelTime in seconds.*	Boolean
GetVisualRayTrace(int pIndex) *Returns all the points in a visual ray trace path as a double array. pIndex is path number.* *Each point has the following order of parameters:* *[x ,0] Initial ray angle (radians)* *[x,1] Range (meters)* *[x ,2] Depth (meters)*	Double[x,3]
GetVisualRayTraceLength(int pIndex) *Returns the length of a visual ray trace path. pIndex is path number.*	Integer
GetVisualRayTracePoint(int pIndex, int pPointNum, out double pInitialAngle, out double pRange, out double pDepth) *Returns a single point in the visual ray trace. pIndex is path number and pPointNum is point number in the path. pInitialAngle is in radians, pRange in meters and pDepth in meters.*	Boolean

Table 4.3 Available values of resultCat with description.

resultCat	Description
0	Transmission loss from transmitter to target
1	Transmission loss from target to receiver
2	Signal excess
3	Probability of detection
4	Total reverberation
5	Surface reverberation
6	Volume reverberation
7	Bottom reverberation
8	Noise after processing
9	Ambient noise
10	Masking level

An example of how some of the result functions can be used is shown below. Three methods are called, returning the masking level, the noise after processing and the parameters used in the calculations.

```
// Get the masking level array
int numberOfValues = _lybin.RangeCells;
double[] maskingLevelValues = new double[numberOfValues];
Object objekt;
_lybin.GetResultsBin(10, out objekt);
maskingLevelValues = (double[])objekt;

//Get noise
double noise;
Object ob;
_lybin.GetResultsBin(8, out ob);
noise = (double)ob;

// Get result model data parameters
string parameters;
_lybin.GetResultModelData(out parameters);
```

Figure 4.1 C# code example: Masking level, noise and model data parameters are collected after the results are calculated.

Code examples

A.1 C# code example

```
using LybinCom;

namespace BrukLybinComEksempel
{
    public partial class Form1 : Form
    {
        // Create
        private readonly LybinModelComBinClass _lybin =
             new LybinModelComBinClass();

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Set the first bottom loss table
            var bl = new double[3, 2];
            bl[0, 0] = 10;
            bl[0, 1] = 40;
            bl[1, 0] = 30;
            bl[1, 1] = 40;
            bl[2, 0] = 56;
            bl[2, 1] = 40;
            _lybin.SetFirstBottomLossTable(0, 4000, bl);

            // Second bottom loss table
            bl[0, 0] = 10;
            bl[0, 1] = 80;
            bl[1, 0] = 30;
            bl[1, 1] = 80;
            bl[2, 0] = 56;
            bl[2, 1] = 80;
            _lybin.AddBottomLossTable(4000, 10000, bl);
            _lybin.UseMeasuredBottomLoss = true;

            // Bottom profile
            var bp = new double[2, 2];
            bp[0, 0] = 0;
            bp[0, 1] = 200;
            bp[1, 0] = 5000;
            bp[1, 1] = 188;
            _lybin.BottomProfile = bp;


            // Set the wave height
```

```
var wh = new double[2, 3];
wh[0, 0] = 0;
wh[0, 1] = 4000;
wh[0, 2] = 5;
wh[1, 1] = 4000;
wh[1, 1] = 9000;
wh[1, 2] = 3;
_lybin.WaveHeight = wh;
_lybin.UseWaveHeight = true;

// Set bottom back scatter table
double[,] bb = new double[3, 2];
bb[0, 0] = 11;
bb[0, 1] = 3.2;
bb[1, 0] = 33;
bb[1, 1] = 7.4;
bb[2, 0] = 88;
bb[2, 1] = 4;
_lybin.SetFirstBottomBackScatterTable(0, 10000, bb);
_lybin.TypeOfRevNoiseCalculation = 1;

// Set volume back scatter profile
double[,] vc = new double[2, 2];
vc[0, 0] = 10;
vc[0, 1] = -80;
vc[1, 0] = 50;
vc[1, 1] = -92;
_lybin.SetFirstVolBackScatterProfile(0, 10000, vc);

// Set the first sound speed profile
// Containing sound speed, temperature and salinity
var ssp = new double[2, 4];
ssp[0, 0] = 0;
ssp[0, 1] = 1480;
ssp[0, 2] = 7;
ssp[0, 3] = 35;
ssp[1, 0] = 620;
ssp[1, 1] = 1510;
ssp[1, 2] = 8;
ssp[1, 3] = 34;
_lybin.SetFirstSoundSpeedTempAndSalinityProfile(0, 2000, ssp);

// Set the second sound speed profile
// Containing only sound speed
var sss = new double[2, 2];
sss[0, 0] = 50;
sss[0, 1] = 1488;
sss[1, 0] = 100;
sss[1, 1] = 1499;
_lybin.AddSoundSpeedProfile(2000, 5000, sss);

// Set the third sound speed profile
// Containing temperature and salinity
var tsp = new double[2, 3];
tsp[0, 0] = 10;
tsp[0, 1] = 6.1;
```

```
tsp[0, 2] = 34;
tsp[1, 0] = 200;
tsp[1, 1] = 4.2;
tsp[1, 2] = 33;
_lybin.AddTempAndSalinityProfile(5000, 8000, tsp);

// Set sonar parameters
_lybin.Depth = 50;
_lybin.TiltReceiver = 0;
_lybin.TiltTransmitter = 0;
_lybin.SideLobeReceiver = 12;
_lybin.SideLobeTransmitter = 12;
_lybin.DetectionThreshold = 13;
_lybin.Frequency = 1000;
_lybin.DirectivityIndex = 25;
_lybin.SourceLevel = 210;
_lybin.BeamWidthReceiver = 30;
_lybin.BeamWidthTransmitter = 18;
_lybin.Length = 1000;
_lybin.FilterBandWidth = 500;
_lybin.Form = "CW";

// Set calculation parameters
_lybin.SetRangeScaleAndRangeCells(10000, 100);
_lybin.SetDepthScaleAndDepthCells(600, 50);
_lybin.TRLRays = 5000;

// Set ocean parameters
_lybin.TargetStrength = 10;
_lybin.ReverberationZone = ReverberationZone.Typical;

 // Let LybinCom calculate noise
_lybin.NoiseCalculation = true;
_lybin.PrecipitationNoiseType = PrecipitationType.LightRain;


// Calculate ray trace for visuaisation
_lybin.VisualRayTraceCalculation = true;
_lybin.VisualSurfaceHits = 6;
_lybin.VisualBottomHits = 8;
_lybin.VisualNumRays = 66;

// Let LybinCom calculate noise
_lybin.NoiseCalculation = true;
_lybin.PrecipitationNoiseType = PrecipitationType.LightRain;


// Do calculation
_lybin.DoCalculation();

// Get raytrace for visualization
int pathCount = _lybin.VisualRayTraceCount;
int travelLength =
       _lybin.GetVisualRayTraceLength(pathCount / 2);
object obj = _lybin.GetVisualRayTrace(pathCount / 2);
```

```
            // Get the ambient noise used
            double used = _lybin.AmbientNoiseLevelUsed;

            // Get calculation results
            string modelData, trl, sig, pod, tot;
            Object mask;

            // XML
            _lybin.GetResults(0, out trl);
            _lybin.GetResults(2, out sig);
            _lybin.GetResults(3, out pod);
            _lybin.GetResults(4, out tot);

            // Binary
            _lybin.GetResultsBin(10, out mask);


            // Get modeldata used in the calculations
            _lybin.GetResultModelData(out modelData);

            // Display in textbox
            textBox1.Text = modelData;
        }

    }
}
```

A.2 Matlab code example

```
% LybinCom used in Matlab %
%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
% initiate LybinCom
lb=actxserver('LybinCom.LybinModelComBin');

% Interfaces
env = lb.invoke('IEnvironment'); % Environment
mod = lb.invoke('IModelData'); % Model
sensor = lb.invoke('ISensor'); % Sonar
pulse = lb.invoke('IPuls'); % Pulse
platform = lb.invoke('IPlatform'); % Platform
ocean = lb.invoke('IOcean'); % Ocean

% Sonar parameters
sensor.Depth = 50;
sensor.TiltReceiver = 0;
sensor.TiltTransmitter = 0;
sensor.SideLobeReceiver = 12;
sensor.SideLobeTransmitter = 12;
sensor.DetectionThreshold = 13;
sensor.Frequency = 1000;
sensor.DirectivityIndex = 25;
```

```
sensor.SourceLevel = 210;
sensor.BeamWidthReceiver = 18;
sensor.BeamWidthTransmitter = 18;

% Pulse
pulse.Length = 1000;
pulse.FMBandWidth = 1000;

%Platform
platform.SelfNoise = 60; % [dB]

% Model
R = 10000;
Z = 600;
R_cells = 100;
Z_cells = 50;
mod.SetRangeScaleAndRangeCells(R, R_cells);
mod.SetDepthScaleAndDepthCells(Z, Z_cells);
mod.TRLRays = 1000;

% Target strength
ocean.TargetStrength = 10;
ocean.PrecipitationNoiseType = 'hail';

% Environment
%%%%%%%%%%%%%%%

% WindSpeed
env.WindSpeedMeasurments = [0,5000,10; 5000,10000,6];

% Sound speed
env.SetFirstSoundSpeedProfile(0, 0, [0, 1480; 660, 1510]);

 % Bottom type
env.BottomType = [0,5000,3;5000,10000,4];

% Bottom profile
prof = [0,450; 2000,470;  4000,450; 5000,480; 7000,550; 10000,580];
env.bottomProfile = prof;

% Calculate
lb.DoCalculation
lb.modelData

% Get calculation results
data.trl.forward = lb.TransmissionLossTransmitter;
data.trl.backward = lb.TransmissionLossReceiver;
data.sig = lb.SignalExcess;
data.pod = lb.ProbabilityOfDetection;
data.rev.Tot_rev = lb.TotalReverberation;
data.rev.Surf_rev = lb.SurfaceReverberation;
data.rev.Vol_rev = lb.VolumeReverberation;
data.rev.Bot_rev = lb.BottomReverberation;
```

```
data.rev.Noise = lb.NoiseAfterProcessing;

% Plot data
maxRange = mod.RangeScale;
maxDepth = mod.DepthScale;
R_cells = mod.RangeCells;
r_plotting = maxRange/R_cells * (0.5:(R_cells-0.5));
z_plotting = maxDepth/Z_cells * (0.5:(Z_cells-0.5));
figure
pcolor(r_plotting,z_plotting,lb.SignalExcess)
shading interp
colormap(jet(13));
set(gca,'CLim',[-25, 40]);
colorbar;
set(gca, 'ydir', 'reverse')
xlabel('Range [m]')
ylabel('Depth [m]')
title('Signal Excess')
hold on
fill([prof(:,1);0;0], [prof(:,2); maxRange; maxRange], 'k');
hold off

% Release interfaces
env.release; % Environment
mod.release; % Model
sensor.release; % Sonar
pulse.release; % Pulse
platform.release; % Platform
ocean.release; % Ocean
lb.release; % Component
```

A.3 C++ example code using LybinCom

```
bool LybinIntegration::Init()
{
   //Instantiate COM object and
   //get access to interface ILybinModelComBin
   m_hr =

m_LybinCom.CoCreateInstance(CComBSTR("LybinCom.LybinModelComBin"
));
   if(FAILED(m_hr))
      return FALSE;

   //Get access to the interfaces in LybinCom
   m_modelData = m_LybinCom;
   m_environment = m_LybinCom;
   m_ocean = m_LybinCom;
   m_platform = m_LybinCom;
```

```
   m_pulse = m_LybinCom;
   m_sensor = m_LybinCom;

   return TRUE;
 }

void LybinIntegration::SetLambertsParameter(double
&lambertsParameter)
{
   //Set type of rev noise calculation to Lamberts rule
   long typeOfRevNoiseCalculation = 3;
   m_hr = m_modelData->
         put_TypeOfRevNoiseCalculation(typeOfRevNoiseCalculation
);

   //Array of Lamberts coefficients (range independent hare)
   CComSafeArray<double> *safeArray;
   double lp[2][3];
   lp[0][0] = 0;
   lp[0][1] = 200;
   lp[0][2] = lambertsParameter;
   lp[1][0] = 200;
   lp[1][1] = 2000;
   lp[1][2] = lambertsParameter;

   // Declare the variable used to store the array indexes
   LONG aIndex[2];

   // Define the array bound structure
   CComSafeArrayBound bound[2];
   bound[0].SetCount(2);
   bound[0].SetLowerBound(0);
   bound[1].SetCount(3);
   bound[1].SetLowerBound(0);

   // Create a new array
   safeArray = new CComSafeArray<double>(bound,2);

   // Use MultiDimSetAt to store doubles in the array
   for (int x = 0; x < 2; x++)
   {
     for (int y = 0; y < 3; y++)
     {
           aIndex[0] = x;
           aIndex[1] = y;
           HRESULT hr = safeArray->
                              MultiDimSetAt(aIndex,lp[x][y]);
           ATLASSERT(hr == S_OK);
     }
   }
```

```
    //Connect to variant
    CComVariant var(*safeArray);
    var.vt = (VT_ARRAY | VT_R8);
    m_hr = m_environment->put_LambertsCoefficient(var);
}

void LybinIntegration::SetRayleighBottomLoss(double
&sed_attenuation, double &sed_rho, double &sed_soundSpeed)
{
    //Tell LybinCom to calculate and use Rayleigh bottom loss
    m_hr = m_modelData->put_UseRayleighBottomLoss(VARIANT_TRUE);

    CComSafeArray<double> *safeArray;

    double rl[1][3];
    rl[0][0] = sed_attenuation;
    rl[0][1] = sed_soundSpeed;
    rl[0][2] = sed_rho;

    // Declare the variable used to store the array indexes
    LONG aIndex[2];

    // Define the array bound structure
    CComSafeArrayBound bound[2];
    bound[0].SetCount(1);
    bound[0].SetLowerBound(0);
    bound[1].SetCount(3);
    bound[1].SetLowerBound(0);

    // Create a new array
    safeArray = new CComSafeArray<double>(bound,2);

    // Use MultiDimSetAt to store doubles in the array
    for (int x = 0; x < 1; x++)
    {
      for (int y = 0; y < 3; y++)
      {
            aIndex[0] = x;
            aIndex[1] = y;
            HRESULT hr = safeArray->
                              MultiDimSetAt(aIndex,rl[x][y]);
            ATLASSERT(hr == S_OK);
      }
    }

    //Connect to variant
    CComVariant var(*safeArray);
    var.vt = (VT_ARRAY | VT_R8);
    m_hr = m_environment->put_RayleighBottomLoss(var);
```

```
}

void LybinIntegration::SetWindSpeed(double &windSpeed)
{
   //Range dependent array of wind speed (not range independent
here)
   CComSafeArray<double> *safeArray;
   double ws[2][3];
   ws[0][0] = 0;
   ws[0][1] = 300;
   ws[0][2] = abs(windSpeed);
   ws[1][0] = 300;
   ws[1][1] = 3000;
   ws[1][2] = abs(windSpeed);

   // Declare the variable used to store the array indexes
   LONG aIndex[2];

   // Define the array bound structure
   CComSafeArrayBound bound[2];
   bound[0].SetCount(2);
   bound[0].SetLowerBound(0);
   bound[1].SetCount(3);
   bound[1].SetLowerBound(0);

   // Create a new array
   safeArray = new CComSafeArray<double>(bound,2);

   // Use MultiDimSetAt to store doubles in the array
   for (int x = 0; x < 2; x++)
   {
     for (int y = 0; y < 3; y++)
     {
           aIndex[0] = x;
           aIndex[1] = y;
           HRESULT hr = safeArray->
                              MultiDimSetAt(aIndex,ws[x][y]);
           ATLASSERT(hr == S_OK);
     }
   }

   //Connect to variant
   CComVariant var(*safeArray);
   var.vt = (VT_ARRAY | VT_R8);
   m_hr = m_environment->put_WindSpeedMeasurments(var);
}

void LybinIntegration::SetWaterDepth(double &waterDepth)
{
   m_hr = m_modelData->put_DepthScale(waterDepth);
```

```
   //Range dependent array of bottom depth (range independent
hare)
   CComSafeArray<double> *safeArray;
   double bb[1][2];
   bb[0][0] = 10;
   bb[0][1] = waterDepth;

   // Declare the variable used to store the array indexes
   LONG aIndex[2];

   // Define the array bound structure
   CComSafeArrayBound bound[2];
   bound[0].SetCount(1);
   bound[0].SetLowerBound(0);
   bound[1].SetCount(2);
   bound[1].SetLowerBound(0);

   // Create a new array
   safeArray = new CComSafeArray<double>(bound,2);

   // Use MultiDimSetAt to store doubles in the array
   for (int y = 0; y < 2; y++)
   {
     aIndex[0] = 0;
     aIndex[1] = y;
     HRESULT hr = safeArray->MultiDimSetAt(aIndex,bb[0][y]);
     ATLASSERT(hr == S_OK);
   }

   //Connect to variant
   CComVariant var(*safeArray);
   var.vt = (VT_ARRAY | VT_R8);
   m_hr = m_environment->put_BottomProfile(var);

}

void LybinIntegration::SetVerticalBeamWidth(double
&verticalBeamWidth)
{
   m_hr = m_sensor->put_BeamWidthTransmitter(verticalBeamWidth);
}

void LybinIntegration::SetDepthTransmitter(double
&transmitterDepth)
{
   m_hr = m_sensor->put_Depth(transmitterDepth);
}

void LybinIntegration::SetPulseLength(double &pulseLength)
```

```
{
   //LybinCom must have pulse length in milli sec
   double pulseLengthMilliSec = pulseLength*1000;
   m_hr = m_pulse->put_Length(pulseLengthMilliSec);
 }

void LybinIntegration::SetFrequency(double &frequency)
{
   m_sensor->put_Frequency(frequency);
}

void LybinIntegration::SetSoundSpeed(int numPoints, double
*depth, double *soundSpeed)
{
   CComSafeArray<double> *safeArray;

   // Define the array bound structure
   CComSafeArrayBound bound[2];
   bound[0].SetCount(2);
   bound[0].SetLowerBound(0);
   bound[1].SetCount(numPoints);
   bound[1].SetLowerBound(0);

   // Create a new array
   safeArray = new CComSafeArray<double>(bound,2);

   // Declare the variable used to store the array indexes
   LONG aIndex[2];

   // Use MultiDimSetAt to store doubles in the array
   for (int x = 0; x < numPoints; x++)
   {
      aIndex[0] = x;
      aIndex[1] = 0;
      HRESULT hr = safeArray->MultiDimSetAt(aIndex,depth[x]);
      ATLASSERT(hr == S_OK);

      aIndex[0] = x;
      aIndex[1] = 1;
      hr = safeArray->MultiDimSetAt(aIndex,soundSpeed[x]);
      ATLASSERT(hr == S_OK);
   }

   //Connect to CComVariant
   CComVariant var(*safeArray);
   var.vt = (VT_ARRAY | VT_R8);

   long pStart = 0;
   long pStop = 3000;
   m_hr = m_environment->
```

```
        raw_SetFirstSoundSpeedProfile(pStart, pStop, var);
}

void LybinIntegration::SetRangeAndRangeCells(double maxTime, int
numOutputPoints)
{
   //Assume sound speed to use in transformation between time
and range
   double soundSpeed = 1500;

   //Calculate max range
   double maxRange = maxTime*soundSpeed/2;

   //Set parameters in LybinCom
   m_hr = m_modelData->
         raw_SetRangeScaleAndRangeCells(maxRange,
numOutputPoints);
}

void LybinIntegration::GetBottomReverberation(int numPoints,
double *revArray)
{
   //Get calculated bottom reverberation from LybinCom
   CComVariant bottomReverberation;
   m_hr = m_ptr->get_BottomReverberation(&bottomReverberation);

   //Unwrap to CComSafeArray
   CComSafeArray<double> safeArray;
   safeArray.Attach(bottomReverberation.parray);

   int numLybinPoints = safeArray.GetCount(0);
   int numRevPoints;
   if(numLybinPoints < numPoints)
     numRevPoints = numLybinPoints;
   else
     numRevPoints = numPoints;

   //Fill in the double arrays with collected values;
   for (int i = 0; i < numrevPoints; i++)
   {
     revArray[i] = safeArray[i];
   }

   safeArray.Detatch();
}

bool LybinIntegration::GetRangeScaleAndCells()
{
   long rangeCells;
   double rangeScale;
```

```
   m_hr = m_modelData->get_RangeCells(&rangeCells);
   m_hr = m_modelData->get_RangeScale(&rangeScale);

   return true;
}

bool LybinIntegration::GetLambertsParameter()
{
   long typeOfRevNoiseCalculation;
   m_hr = m_modelData->
         get_TypeOfRevNoiseCalculation(&typeOfRevNoiseCalculatio
n);

   VARIANT lambertsParameter;
   m_hr = m_environment-
>get_LambertsCoefficient(&lambertsParameter);

   return true;
}

bool LybinIntegration::GetRayleighBottomLossParameters()
{
   VARIANT_BOOL use;
   m_hr = m_modelData->get_UseRayleighBottomLoss(&use);

   VARIANT rayleigh;
   m_hr = m_environment->get_RayleighBottomLoss(&rayleigh);

   return true;
}

bool LybinIntegration::GetWindSpeed()
{
   VARIANT windSpeed;
   m_hr = m_environment->get_WindSpeedMeasurments(&windSpeed);

   return true;
}

bool LybinIntegration::GetSoundSpeed()
{
   long pIndex = 0;
   long pStart;
   long pStop;
   VARIANT pProfile;

   m_hr = m_environment->
   raw_GetSoundSpeedProfile(pIndex, &pStart, &pStop, &pProfile);

   return true;
```

```
}

double LybinIntegration::GetWaterDepth()
{
   double waterDepth;
   m_hr = m_modelData->get_DepthScale(&waterDepth);

   return waterDepth;
}

double LybinIntegration::GetVerticalBeamWidth()
{
   double verticalBeamWidth;
   m_hr = m_sensor-
>get_BeamWidthTransmitter(&verticalBeamWidth);

   return verticalBeamWidth;
}

double LybinIntegration::GetDepthTransmitter()
{
   double transmitterDepth;
   m_hr = m_sensor->get_Depth(&transmitterDepth);

   return transmitterDepth;
}

double LybinIntegration::GetPulseLength()
{
   double pulseLength;
   m_hr = m_pulse->get_Length(&pulseLength);

   return pulseLength;
}

double LybinIntegration::GetFrequency()
{
   double frequency;
   m_hr = m_sensor->get_Frequency(&frequency);

   return frequency;
}

bool LybinIntegration::CleanUp()
{
   //Cleanup
   if(m_modelData)
     m_modelData.Release();
   if(m_environment)
     m_environment.Release();
```

```
    if(m_pulse)
      m_pulse.Release();
    if(m_sensor)
      m_sensor.Release();
    if(m_LybinCom)
      m_LybinCom.Release();

    return true;
}
```

Abbreviations

FFI	Norwegian Defense Research Institute
NDLO	Norwegian Defense Logistic Organization
GFA	Government Furnished Assets
LYBIN	LYdBane og INtensitetsprogram (acoustic model)
XML	Extensible Markup Language
COM	Component Object Model

Type definitions

Integer	32-bit integer
Double	64-bit floating point
Boolean	16-bit integer (0: false, 1: true)
String	BSTR. Basic string used by COM
Enum	32-bit integer
TravelTimePoint	256-bit struct defined in LybinCom Type library

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. Bøhler, "LybinTCPserver 7.0.5 - Interface description," FFI-Note 2023/02442, 2023.

7. https://docs.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal

8. https://thrift.apache.org/

About FFI
The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's mission
FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's vision
FFI turns knowledge and ideas into an efficient defence.

FFI's characteristics
Creative, daring, broad-minded and responsible.