FFI-rapport 2012/01453

LybinCom 6.1 – description of the binary interface

Elin Dombestein

Norwegian Defence Research Establishment (FFI)

24 August 2012

FFI-rapport 2012/01453

366401

P: ISBN 978-82-464-2123-0
E: ISBN 978-82-464-2124-7

Keywords

Akustisk deteksjon

Modellering og simulering

Programmering (Databehandling)

Approved by

Connie Elise Solberg	Project Manager
Elling Tveit	Director

English summary

The acoustic ray trace model LYBIN is a well established and frequently used sonar prediction model owned by the Norwegian Defence Logistic Organisation. The model is used aboard navy vessels as well as in training situations on shore. LYBIN has become an important tool in both planning and evaluation of maritime operations, and earlier versions are already integrated in combat system software, tactical decision aids and tactical trainers.

The calculation kernel of LYBIN is implemented as a software module called LybinCom. In addition there exists a graphical user interface which can be used together with LybinCom to build a stand alone executive application. We call this stand alone executive application LYBIN.

An implementation as a software module makes LybinCom suitable for integration with other applications, and enables LybinCom to interact with other mathematical models, web services, geographical information systems and more. Third parties can integrate LybinCom in their software without needing access to the source code.

LybinCom has two different interfaces for data exchange with other software. The two interfaces are the binary interface and the e**X**tensible **M**arkup **L**anguage (XML) interface. The binary interface enables fast transportation of large amounts of data to and from LybinCom. The XML interface is not as fast, but is more robust because the format of the input files is not as strict. The XML interface discards any parts of the input file it does not recognize.

This report describes the binary software interface of LybinCom 6.1 needed for the integration of LybinCom with other software applications. All parameters and data sets that can be passed to and from LybinCom are described. Examples of programming code for integration of LybinCom are also included. The interface's inner structure and how the acoustic modelling is performed are not described in this report.

Sammendrag

Den akustiske strålegangsmodellen LYBIN er en etablert og mye brukt sonar ytelsesmodell som eies av Forsvarets Logistikkorganisasjon. Modellen brukes både ombord på marinefartøy og i treningssituasjoner på land. LYBIN er blitt et viktig verktøy både i planlegging og evaluering av maritime operasjoner, og tidligere versjoner er allerede integrert i programvare for kampsystemer, taktisk beslutningsstøtte og taktiske trenere.

LYBINs beregningskjerne er implementert som en software modul kalt LybinCom. I tillegg eksisterer det et grafisk brukergrensesnitt som sammen med LybinCom kan brukes for å bygge en frittstående eksekverbar applikasjon. Vi kaller denne frittstående applikasjonen for LYBIN.

Implementasjonen som en software modul gjør LybinCom egnet for integrasjon med andre applikasjoner, og muliggjør at LybinCom kan samhandle med andre matematiske modeller, web-tjenester, geografiske informasjonssystemer med mer. Det er mulig for andre å integrere LybinCom i deres programvare uten å ha tilgang til kildekoden.

LybinCom har to ulike grensesnitt for datautveksling med annen programvare. De to grensesnittene er det binære grensesnittet og e**X**tensible **M**arkup **L**anguage (XML) grensesnittet. Det binære grensesnittet muliggjør rask transport av store mengder data til og fra LybinCom. XML grensesnittet er ikke like raskt, men er mer robust fordi formatet til inputfilene ikke er så rigid. XML grensesnittet forkaster de delene av inputfila det ikke gjenkjenner.

Denne rapporten beskriver det binære grensesnittet til LybinCom 6.1 som trengs for å kunne integrere LybinCom med andre programvareapplikasjoner. Alle parametere og datasett som kan sendes til og fra LybinCom er beskrevet i denne rapporten. Noen eksempler på programkode for integrasjon av LybinCom er også inkludert. Det som skjer innenfor grensesnittene og hvordan den akustiske modelleringen er gjort vil ikke bli omtalt i denne rapporten.

Contents

1 Introduction 7

2 Description of LybinCom 9
2.1 The data layer 10
2.1.1 Range dependent input data 12
2.2 The calculation layer 13
2.3 The result layer 14

3 Input data 14
3.1 LybinModelData class 14
3.1.1 Switches 19
3.2 Environment class 20
3.3 Ocean class 20
3.4 WindSpeedMeasurement class 21
3.5 WaveHeight class 22
3.6 SoundSpeed class 23
3.7 BottomProfile class 26
3.8 BottomType class 27
3.9 BottomLoss class 28
3.10 Rayleigh bottom loss 29
3.11 BottomBackScatter class 30
3.12 LambertsCoefficient class 32
3.13 VolumeBackScatter class 33
3.14 ReverberationAndNoiseMeasurements class 35
3.15 Platform class 35
3.16 Sensor class 36
3.17 Pulse class 37

4 Initiate calculation 38

5 Calculation results 38

Appendix A Example code 46
A.1 C# Windows forms application using LybinCom 46
A.2 Matlab file using LybinCom, basic example 49
A.3 Matlab file using LybinCom, impulse response example 51
A.4 C++ example code using LybinCom 52

References 60

Abbreviations 61

Definitions 61

1 Introduction

The acoustic ray trace model LYBIN is a well established and frequently used sonar prediction model owned by the Norwegian Defence Logistic Organisation (NDLO). The model is used aboard navy vessels as well as in training situations on shore. LYBIN has become an important tool in both planning and evaluation of maritime operations.

LYBIN is a range dependent two-dimensional model. Several thousand 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 strength, the calculated transmission loss, reverberation and noise. LYBIN is a robust, user friendly and fast acoustic ray-trace simulator. The physical foundations for the model are described in more detail in [1] and [2].

On behalf of NDLO, the Norwegian Defence Research Institute (FFI) has been responsible for testing, evaluation and further development of LYBIN since the year 2000. During this period, several new versions of LYBIN have been released. LYBIN 6.1 was released in august 2012.

LYBIN 6.1 is divided into two separate parts: the calculation kernel and the graphical user interface. This separation enables LYBIN to interact with other applications, such as mathematical models, web services, geographic information systems, and more.

The graphical user interface represents the classical LYBIN application, where LYBIN is used as stand-alone software. Environmental data and information about the sonar and sonar platform are sent to the calculation kernel by the operator through the graphical user interface. The calculation results are thereafter displayed by the graphical user interface. A picture of LYBIN 6.1 graphical user interface is shown in Figure 1.1.

The stand-alone calculation kernel, called LybinCom 6.1, enhances the potential applicability of LYBIN by enabling connectivity and communication between systems. LybinCom can be integrated with external applications, and both input and calculation results can be handled automatically from outside applications.

One example of integration of LybinCom is an application displaying sonar coverage. The environmental data are fed into LybinCom and calculations started from the external application, while the calculation results are displayed in the external program. The output from such an application is shown in Figure 1.2, where calculated signal excess is plotted in the geographical information system Maria [3]. The integration between LybinCom and Maria is described in more detail in [4] and [5].

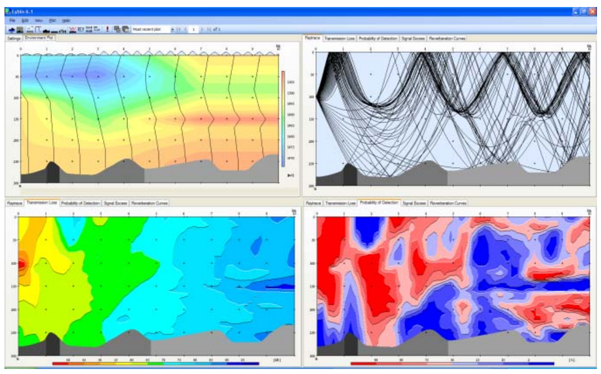
Figure 1.1 LYBIN 6.1 graphical user interface.

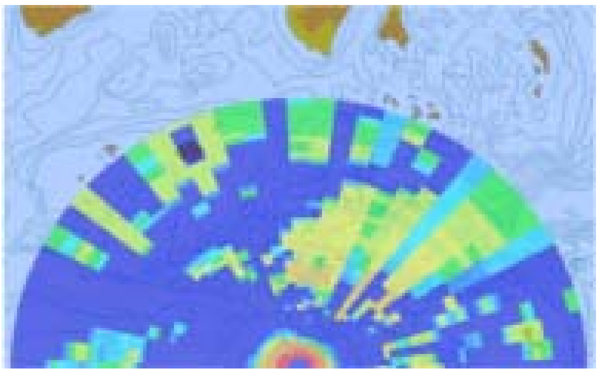
Figure 1.2 LybinCom integrated with the geographical information system maria

The **C**omponent **O**bject **M**odel (COM) called LybinCom has two different interfaces for data exchange. The two interfaces are the *binary interface* and the *eXtensible Markup Language* (XML) interface. The XML interface uses e**X**tensible **M**arkup **L**anguage (XML) to control and manage the information going to and from LYBIN. XML is an open standard with a simple syntax and an unambiguous structure. The XML interface is very robust since it discards any parts of the code it does not recognize.

The binary interface provides a faster data exchange than the XML interface. The binary file format is more rigorous though. Deviations from the defined format will lead to failures in the data transfer process. The binary interface contains both method calls and variable calls to access the kernel. The binary interface of LybinCom offers more functionality than the XML interface. In fact, the XML interface can be called through the binary interface of LybinCom as a method call.

LybinCom is implemented as a COM module for the windows platform. COM is a language-neutral way of implementing objects that can be used in environments different from the one they where created in. LybinCom is implemented in C++, but is available from software languages like C#, C++ and Matlab. LybinCom has a series of interfaces, all with their properties and functions available. The interfaces are listed in Table 1.1.

Interface name	Description
IEnvironment	Environmental data.
ILybinModelCom	Start calculation, get results and see the calculation parameters in XML form.
ILybinModelComBin	Start calculation, get results and see the calculation parameters in binary form.
IModelData	Parameters controlling the calculations such as accuracy, which datasets to be used etc.
IOcean	Parameters describing the media (ocean) and the assumed target.
IPlatform	Description of the platform holding the sensor.
IPulse	Parameters describing the sensor pulse.
ISensor	Parameters describing the sensor.

Table 1.1 LybinCom interfaces.

By using a .NET language such as C#.NET, C++.NET, Visual Basic .NET, Fortran.NET, Pyton.NET etc, classes defined in the COM Type Library is visible. All the code examples in this report are written in C#.NET using the object LybinModelComBinClass. LybinModelComBinClass contain all interfaces, and is thus an alternative to using interfaces directly. LybinModelComClass is a similar class containing only the interface concerning XML, and is thus equal to the ILybinModelCom interface found in Table 1.1.

Using unmanaged languages, like traditional C++ or Matlab, the classes defined in the COM Type Library is not visible. Implementation using these languages is done using the interfaces described in Table 1.1.

This document describes the binary interface of LybinCom 6.1. Documentation of LYBINs earlier software interfaces can be found in [6], [7], [8] and [9].

2 Description of LybinCom

LybinCom is the calculation engine of the total LYBIN software package. LybinCom process the environmental and sonar information, and perform all the mathematical calculations. If we take a look inside LybinCom, we will see that it is divided into three separate parts. We have the data layer, the calculation layer and the result layer. Each of these is described in more detail below. A schematic description of LybinCom is shown in Figure 2.1. The arrows indicate the data flow in-between the different calculation stages (boxes). The calculation results from one box are used in another of the arrow is pointing to the current box.

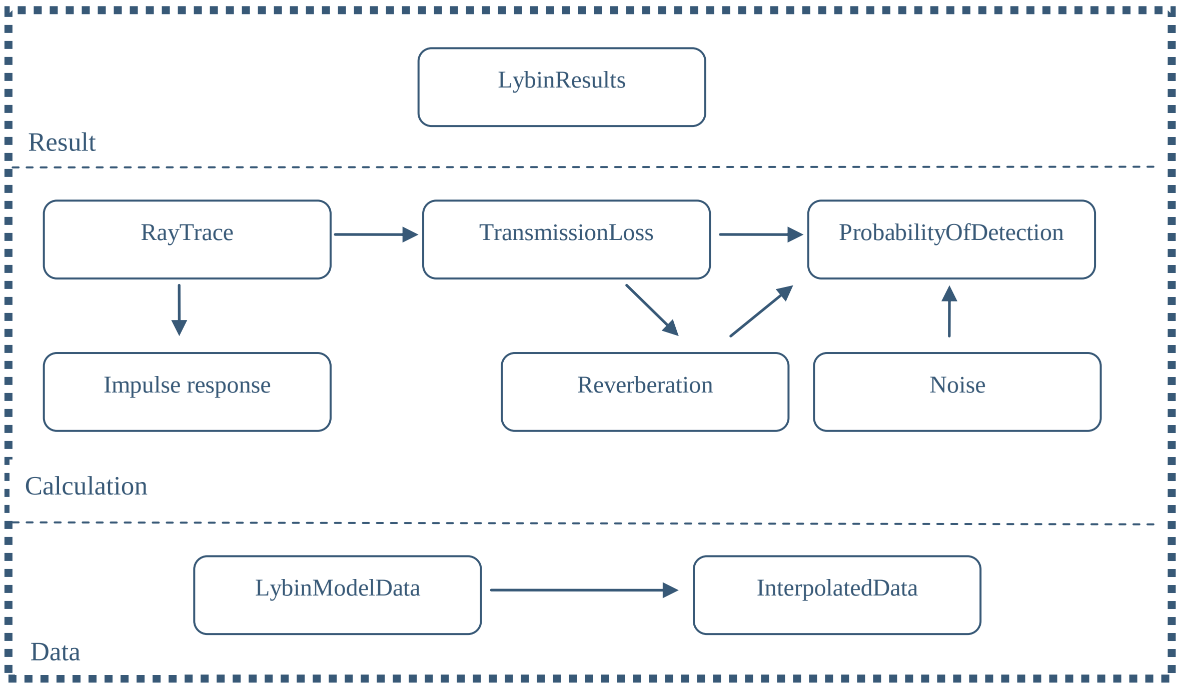

Figure 2.1 Schematic description of LybinCom.

2.1 The data layer

The data layer handles all the data given to the model. All the datasets forming the basis of the acoustic calculations in LybinCom are organized in classes. A class diagram for the LybinCom input data model is showed in Figure 2.2 and each of these classes are described shortly in Table 2.1.

LybinModelData has two members: the environment class and the platform class. The environment class is an assembly of all the environmental data LYBIN uses in the calculations. The platform class holds all the information about the sonar and the sonar platform.

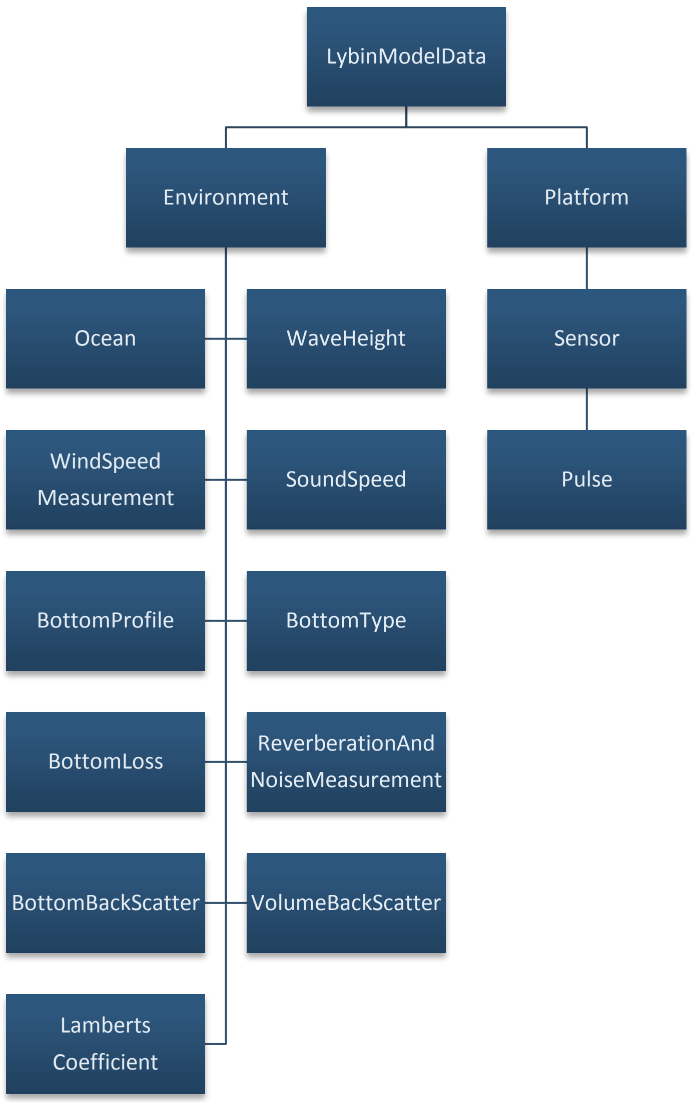

Figure 2.2 The LybinCom input data model (Class diagram).

Class name	Description
BottomBackScatter	Range dependent bottom back scatter values as a function of each ray's grazing angle with the bottom.
BottomLoss	Range dependent bottom loss values as a function of each ray's grazing angle with the bottom.
BottomProfile	Single measurements 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.
Environment	An assembly class for all environmental data: holds no parameters of its own.
LambertsCoefficient	Range dependent bottom back scatter model according to Lamberts law.

Class name	Description
Model	Parameters controlling the calculations such as accuracy, which datasets to be used etc.
Ocean	Parameters describing the media (ocean) and the assumed target, such as ambient noise, pH, surface scatter, target strength and ship density.
Platform	Description of the platform holding the sensor. Contains platform speed and noise.
Pulse	Parameters describing the sensor pulse: its form, length, bandwidth etc.
ReverberationAndNoiseMeasurement	Range dependent total reverberation and noise data.
Sensor	Parameters describing the sensor (sonar) such as depth, tilt, frequency etc.
SoundSpeed	Range dependent sound speed, temperature and salinity measurements as function of depth.
VolumeBackScatter	Range dependent volume back scatter values.
WaveHeight	Range dependent wave height measurements.
WindSpeedMeasurement	Range dependent wind speed measurements.

Table 2.1 LybinCom classes containing input data.

2.1.1 Range dependent input data

LybinCom is able to handle range dependent environments. In LybinCom, range dependent environmental data is specified for certain range intervals from the sonar.

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.

Except for BottomProfile and ReverberationAndNoiseMeasurement, the range dependent data are given with start and stop values to indicate their range of validity. In this context, we call these datasets, with start and stop related to a value (or sets of values), for 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 2.3*. The possible numbers of values to be used in the calculation are only limited by the calculation accuracy.

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 combination of these are possible to use.

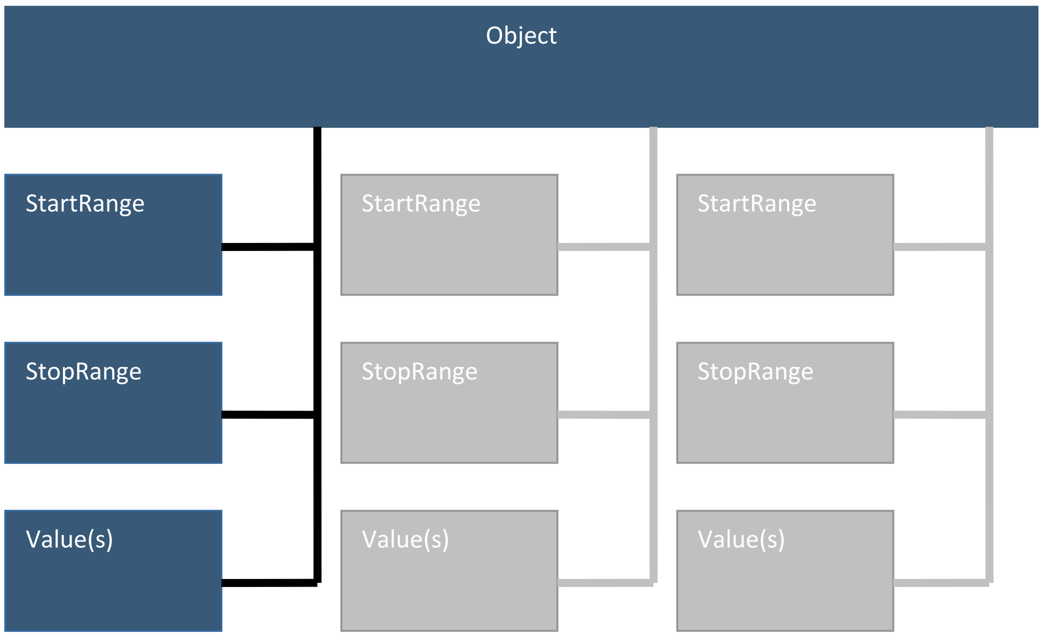

Figure 2.3 Schematic description of a range dependent object with start and stop parameters.

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.

2.2 The calculation layer

In the calculation layer, all the acoustic calculations are performed. First the ray trace is calculated. The intensity of all the rays is then summed up within every calculation cell in order to compute the transmission loss. The reverberation is found based on the backscattering properties and the transmission loss at the sea surface, bottom and volume. Noise is calculated as the sum of the ambient noise in the sea and the sonar self noise. Finally the probability of detection is calculated based on target echo strength, detection threshold, transmission loss, reverberation and noise.

Impulse response is calculated directly from the ray trace. The intensity loss and travel time of all points in all rays are sorted according to travel path history. A travel path history, sometimes called ray family, is a unique sequence of the following: surface reflection, bottom reflection, upper turning point or lower turning point.

2.3 The result layer

The calculation results are managed by the result layer. All the data sets in the result layer are listed below:

- Simple ray trace
- Travel time
- Transmission loss from sonar to target
- Transmission loss from target to sonar
- Impulse response
- Noise
- Surface reverberation
- Volume reverberation
- Bottom reverberation
- Total reverberation
- Masking level
- Signal excess
- Probability of detection

3 Input data

Every class below LybinModelData, as shown in Figure 2.2 is discussed in this section. The access methods and variables are described. Some code examples are also included.

3.1 LybinModelData class

The LybinModelData class contains parameters controlling the acoustic calculations: the resolution of the calculation, what type of calculation to be performed, and so on. All the parameters in LybinModelData are listed in Table 3.1, and the access functions connected to the LybinModelData are described in Table 3.2.

Parameter	Type	Default value	Unit
BottomReverberationCalculation *Switch to control whether to calculate the bottom reverberation or not. The switch is available on the interface but not yet implemented, i.e. will always be set to "true".* *False: This choise is currently not available.* *True: Do calculate bottom reverberation.*	Boolean	true	
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	Meters
DepthScale *Maximum depth in the calculation.*	Double	300	Meters
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	Meters
ImpulseResponseCalculation *Switch to control whether to calculate impulse response or not.* *False: Do not calculate impulse response.* *True: Calculate impulse response.*	Boolean	false	
ImpulseResponseDepth *The depth that the impulse response will be calculated from.*	Double	0	Meters
MaxBorderHits *Maximum number of boundary hits (sea or bottom) allowed before a ray is terminated.*	Integer	5000	
ModelData *Total data model holding all the input parameters to be used in the calculation. The model data are returned as an XML string.*	String (utf-8)		

Parameter	Type	Default value	Unit
NoiseCalculation *Switch to control whether to calculate the noise or not.* *False: Do not calculate noise.* *True: Do calculate noise.*	Boolean	true	
PassiveCalculation *Switch to control whether to perform calculations for active or passive sonar.* *False: Calculate for active sonar.* *True: Calculate for passive sonar.*	Boolean	false	
ProbabilityOfDetectionCalculation *Switch to control whether to calculate the probability of detection or not. The switch is available on the interface but not yet implemented, i.e. will always be set to "true".* *False: This choise is currently not available.* *True: Do calculate the probability of detection.*	Boolean	true	
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	Meters
RangeScale *Maximum range in the calculation.*	Double	10000	Meters
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	Meters
RayTraceCalculation *Switch to control whether to calculate the total ray trace or not. The switch is available on the interface but not yet implemented,*	Boolean	true	

Parameter	Type	Default value	Unit
i.e. will always be set to “true”. *False: This choice is currently not available.* *True: Calculate the total ray trace.*			
SignalExcessCalculation *Switch to control whether to calculate the signal excess or not. The switch is available on the interface but not yet implemented, i.e. will always be set to “true”.* *False: This choice is currently not available.* *True: Calculate the signal excess.*	Boolean	true	
SignalExcessConstant *Parameter affecting the relation between signal excess and probability of detection.*	Double	3	
SurfaceReverberationCalculation *Switch to control whether to calculate the surface reverberation or not. The switch is available on the interface but not yet implemented, i.e. will always be set to “true”.* *False: This choice is currently not available.* *True: Do calculate surface reverberation.*	Boolean	true	
TerminationIntensity *Each ray is terminated when its intensity falls below this value.*	Double	1E-16	
TransmissionLossFromTargetCalculation *Switch to control whether to calculate the transmission loss from target to sonar or not. The switch is available on the interface but not yet implemented, i.e. will always be set to “true”.* *False: This choice is currently not available.* *True: Do calculate the transmission loss from target to sonar.*	Boolean	true	
TransmissionLossToTargetCalculation *Switch to control whether to calculate the transmission loss from sonar to target or not. The switch is available on the interface but not yet implemented, i.e. will always be set to “true”.* *False: This choice is currently not available.* *True: Do calculate the transmission loss from sonar to target.*	Boolean	true	
TravelTimeAngleRes *The distance in degrees between the start angles of the rays to be used in the travel time calculation.*	Double	1	Degrees
TravelTimeCalculation *Switch to control whether to calculate travel time or not.* *False: Do not calculate travel time.* *True: Calculate travel time.*	Boolean	false	

Parameter	Type	Default value	Unit
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*	Integer	0	
UseMeasuredBottomLoss *Tells the model how to calculate bottom loss. If UseRayleighBottomLoss = true, it will overrule UseMeasuredBottomLoss.* *False: Use bottom types to calculate bottom loss* *True: Use measured or supplied bottom loss values*	Boolean	false	
UseMeasuredHorizontalBeamWidth *Tells whether to use the input parameter BeamWidthHorizontal in stead of calculating the horizontal beam.* *False: Use directivity index and vertical beam width to calculate horizontal beam width* *True: Use measured or supplied horizontal beam width*	Boolean	false	
UseRayleighBottomLoss *Tells the model how to calculate bottom loss. If UseRayleighBottomLoss = true, it will overrule UseMeasuredBottomLoss.* *False: Use Rayleigh bottom loss* *True: Do not use Rayleigh bottom loss.*	Boolean	false	
UseWaveHeight *Tells the model to use wave height instead of wind speed.* *False: Use wind speed.* *True: Use wave height.*	Boolean	false	
VisualRayTraceCalculation *Switch to control whether to calculate a ray trace plot for visualisation or not.* *False: Do not calculate ray trace for visualisation.* *True: Calculate ray trace for visualisation.*	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	

Parameter	Type	Default value	Unit
VisualSurfaceHits *Number of surface hits alloved in the visual ray trace plot.*	Integer	2	
VolumeReverberationCalculation *Switch to control whether to calculate the volume reverberation or not. The switch is available on the interface but not yet implemented, i.e. will always be set to "true".* *False: This choise is currently not available.* *True: Do calculate volume reverberation.*	Boolean	true	

Table 3.1 Parameters in the LybinModelData class.

3.1.1 Switches

TypeOfRevNoiseCalculation, UseMeasuredBottomLoss and UseRayleighBottomLoss can make LybinCom use certain datasets instead of predefined default values. In order to follow these demands, the spesified datasets must be sent into LybinCom. If LybinCom cannot find these datasets, the switches will be set back to default values. For all the three parameters TypeOfRevNoiseCalculation, UseMeasuredBottomLoss and UseRayleighBottomLoss, default values mean using the predefined bottom types to calculate bottom reverberation and bottom loss respectively.

Both UseMeasuredBottomLoss and UseRayleighBottomLosss tells LybinCom how to calculate the bottom loss. These two parameters can cause a conflict. If both are set to true, RayleighBottomLoss wil be used.

The interface has various calculation switches that will give clients accessing LybinCom the possibility to decide what data to calculate. These switches are available, but will not affect the returned results from the calculation, i.e. they are all set to "true". The intention is to implement these switches further in a future release. The calculation switches are:

- BottomReverberationCalculation
- ProbablityOfDetectionCalculation
- RayTraceCalculation
- SignalExcessCalculation
- SurfaceReverberationCalculation
- TransmissionLossFromTargetCalculation
- TransmissionLossToTargetCalculation
- VolumeReverberationCalculation

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
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

Table 3.2 Functions in the LybinModelData class.

3.2 Environment class

The environment class does not hold any functions or parameters of its own. It is only an assembly class for all the classes holding environmental data.

3.3 Ocean class

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 3.3. There is no access functions connected to the ocean class.

Ambient noise can either be given as a fixed parameter, AmbientNoiseLevel, or it can be calculated from the given environmental input. Which one of these alternatives to be used is decided by the parameter NoiseCalculation 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 *True: Surface reflected ray angles will be modified in order to simulate rough sea scattering.* *False: Rays hitting the sea surface will be reflected specularly, as from a perfectly smooth surface.*	Boolean	true	
TargetStrength *Target echo strenght.*	Double	10	dB

Table 3.3 Parameters in the Ocean class.

3.4 WindSpeedMeasurement class

The WindspeedMeasurement class only has one accessible parameter, WindSpeedMeasurements, which is listed in Table 3.4.

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

Table 3.4 Parameters in the WindSpeedMeasurement class.

An example of how WindSpeedMeasurements can be used is shown in the C# code example below. In the example, the measured wind speed is 2 meters/second from 0 to 5 kilometers, and 4 meters/second from 5 to 10 kilometers.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Wind
double[,] ws = new double[2, 3];
ws[0, 0] = 0;      // Start
ws[0, 1] = 5000;   // Stop
ws[0, 2] = 2;      // Wind speed
ws[1, 0] = 5000;   // Start
ws[1, 1] = 10000;  // Stop
ws[1, 2] = 4;      // Wind speed

Lybin.WindSpeedMeasurments = ws;
```

3.5 WaveHeight class

The WaveHeight class only has one accessible parameter, the WaveHeight, which is listed in Table 3.5.

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

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

Table 3.5 Parameters in the WaveHeight class.

An example of how WaveHeight can be used is shown below. In the example the wave height is 1 meter from 0 to 5 kilometers, and 2 meters from 5 to 10 kilometers.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Wave height
double[,] wh = new double[2, 3];
wh[0, 0] = 0;      // Start
wh[0, 1] = 5000;   // Stop
wh[0, 2] = 1;      // Wave height
wh[1, 0] = 5000;   // Start
wh[1, 1] = 10000;  // Stop
wh[1, 2] = 2;      // Wave height

Lybin.WaveHeight = wh;
Lybin.UseWaveHeight = true;
```

3.6 SoundSpeed class

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 3.6. 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.

Parameter	Default value	Unit
SoundSpeed	1480	m/s
Temperature	7,36	°C
Salinity	35	parts per thousand

Table 3.6 Default values for profile parameters in the SoundSpeed class.

There is only one parameter in the SoundSpeed class, the SoundSpeedProfileCount, given in Table 3.7. The functions in the SoundSpeed class are given in Table 3.8. 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

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

Table 3.7 Parameters 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**: meters **stop**: meters **<u>profile:</u>** **depth**: meters **sound speed**: meters/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	
AddTempProfile(int start, int stop, object profile) *Add another temperature profile. This function can only be used after*	Void	

Function	Type	Unit of input parameters
the first profile has been added with one of the SetFirstProfile functions.		
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	

Table 3.8 Functions in the SoundSpeed class.

An example of how some of the sound speed functions can be used is shown below. 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 meters depth and the sound speed 1510 m/s, temperature 8° Celsius and a salinity of 34 ppt at 620 meters 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.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// 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);
```

3.7 BottomProfile class

The BottomProfile class only has one accessible parameter, the BottomProfile, which is listed in Table 3.9. The BottomProfile can consist of any number points in range with their corresponding bottom depths.

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

Table 3.9 Parameter in the bottom profile class.

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

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Bottom
double[,] bp = new double[2, 2];
bp[0, 0] = 0;     // Range
bp[0, 1] = 300;   // Depth
bp[1, 0] = 1000; // Range
bp[1, 1] = 380;   // Depth

Lybin.BottomProfile = bp;
```

3.8 BottomType class

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. UseMeasuredBottomLoss and UseRayleighBottomLoss are found in the LybinModelData class.

The BottomType class only has one accessible parameter, BottomType, which is listed in Table 3.10.

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

Table 3.10 Parameters in the BottomType class.

An example of how BottomType can be used is shown below. 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.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Bottom type
double[,] bt = new double[2, 3];
bt[0, 0] = 0;      // Start
bt[0, 1] = 5000;   // Stop
bt[0, 2] = 4;      // Bottom type
bt[1, 0] = 5000;   // Start
bt[1, 1] = 10000;  // Stop
bt[1, 2] = 2.3;   // Bottom type

Lybin.BottomType = bt;
```

3.9 BottomLoss class

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. Bottom loss is generally a function of bottom type, gracing angle and frequency. 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 equidistantly spaced grazing 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 UseMeasuredBottomLoss and UseRayleighBottomLoss can be found in the LybinModelData class.

There is only one parameter in the BottomLoss class, the BottomLossTableCount, given in Table 3.11. The functions in the BottomLoss class are given in Table 3.12.

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

Table 3.11 Parameters 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**: meters, **stop**: meters **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	

Table 3.12 Functions in the BottomLoss class.

An example of how some of the bottom loss functions can be used is shown below. In the example, the first bottom loss fan is set to be valid from 0 km to 30 km. The loss table consist of the following data: 10°= 4.2 dB, 30° = 6.4 dB and 80° = 9 dB. At the end of the example, the first bottom loss table is fetched back from LybinCom.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Set the first bottom loss table
double[,] bl = new double[3, 2];
bl[0, 0] = 10;  // Grazing angle
bl[0, 1] = 4.2; // Bottom loss
bl[1, 0] = 30;  // Grazing angle
bl[1, 1] = 6.4; // Bottom loss
bl[2, 0] = 80;  // Grazing angle
bl[2, 1] = 9;   // Bottom loss
Lybin.SetFirstBottomLossTable(0, 30000, bl);
Lybin.UseMeasuredBottomLoss = true;
Lybin.UseRayleighBottomLoss = false;

// Get the first bottom loss table
int index = 0;
int start, stop;
object table = new object();
Lybin.GetBottomLossTable(index, out start, out stop, out table);
```

3.10 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.

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 water and density in the bottom)*	dB/wavelength, Meters/second , scalar

Table 3.13Parameters in the RayleighBottomLoss class.

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.

An example of how the Rayleigh bottom loss is used is shown in the C# code example below:

```
Lybin = new LybinCom.LybinModelComBinClass();

// Rayleigh parameters
double[,] rbl = new double[1, 3];
rbl[0, 0] = 0.92; // BottomAttenuation;
rbl[0, 1] = 1717; // BottomSoundSpeed;
rbl[0, 2] = 1.81; // DensityRatio;
Lybin.RayleighBottomLoss = rbl;
Lybin.UseRayleighBottomLoss = true;
```

3.11 BottomBackScatter class

Bottom back scatter is the fraction of energy that is scattered back towards to the receiver when a ray hits the sea bottom. The bottom back scattering is generally a function of bottom type, grazing angle and frequency. 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 tabulated values to create backscattering coefficients for equidistantly spaced grazing angles. The back scattering coefficients are given as dB per square meter.

Bottom back scatter is an optional choice 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 3.14. The functions in BottomBackScatter class are given in Table 3.15.

An example of how the some of the bottom back scatter functions can be used is shown in the code example below. In the example, the first bottom back scatter table is set. At the range from 0 km to 30 km LybinCom shall use the data points: 10° = 35 dB, 30° = 25 dB and 80° = 23 dB. At the end of the example, the first bottom back scatter table is fetched back from LybinCom.

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

Table 3.14 Parameters 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**: meters, **stop**: meters **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	

Table 3.15 Functions in the BottomBackScatter class.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Set the first bottom back scatter table
double[,] bc = new double[3, 2];
bc[0, 0] = 10; // Grazing angle
bc[0, 1] = 35; // Back scatter
bc[1, 0] = 30; // Grazing angle
bc[1, 1] = 25; // Back scatter
bc[2, 0] = 80; // Grazing angle
bc[2, 1] = 23; // Back scatter
Lybin.SetFirstBottomBackScatterTable(0, 30000, bc);
Lybin.TypeOfRevNoiseCalculation = 1;

// Get the first bottom back scatter table
int index = 0;
int start, stop;
object table = new object();
Lybin.GetBottomBackScatterTable(index, out start, out stop, out table);
```

3.12 LambertsCoefficient class

Lamberts rule is a possible choice in order to calculate the bottom back scattering coefficients. According to Lamberts rule, the back scattering coefficient is given by [10]:

$$\sigma(\theta) = \mu \sin^2 \theta \tag{3.1}$$

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

The LambertsCoefficient class only has one accessible parameter, the LambertsCoefficient, which is listed in Table 3.16.

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)*	(Meters, Meters, dB)

Table 3.16 Parameters in the LambertsCoefficient class.

An example of how LambertsCoefficient can be used is shown below. In the example the LambertsCoefficient is -20 dB from 0 to 5 kilometers, and -27 dB from 5 to 10 kilometers.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// 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;
```

3.13 VolumeBackScatter class

Volume back scatter is 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 3.17. The functions in the VolumeBackScatter class are given in Table 3.18.

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

Table 3.17 Parameters 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 SetFirstVolumeBackScatterFan 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	

Table 3.18 Functions in the VolumeBackScatter class.

An example of how some of the volume back scatter functions can be used is shown below. In the example, the first volume back scatter profile is set. At the range from 0 km to 10 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.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Set the first volume back scatter profile
double[,] vc = new double[2, 2];
vc[0, 0] = 10;  // Depth
vc[0, 1] = -80; // Back scatter
vc[1, 0] = 50;  // Depth
vc[1, 1] = -92; // Back scatter
Lybin.SetFirstVolBackScatterProfile(0, 10000, 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);
```

3.14 ReverberationAndNoiseMeasurements class

The ReverberationAndNoiseMeasurements class only has one accessible parameter, ReverberationAndNoiseMeasurements, which is listed in Table 3.19. 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 measured reverberation and noise data).

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

Table 3.19 Parameter in the ReverberationAndNoiseMeasurements class.

An example on how the ReverberationAndNoiseMeasurements can be used is shown below. In the example, two points are inserted. The first is the value 80 dB at a range of 2 km. The second is the value 70 dB at a range of 8 km.

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

// Reverberation and noise measurements
double[,] ran = new double[2, 2];
ran[0, 0] = 2000; // Range
ran[0, 1] = 80;   // Measurement
ran[1, 0] = 8000; // Range
ran[1, 1] = 70;   // Measurement

Lybin.ReverberationAndNoiseMeasurements = ran;
Lybin.TypeOfRevNoiseCalculation = 2;
```

3.15 Platform class

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 a helicopter or a buoy. The parameters in the platform class are listed in Table 3.20.

Parameter	Type	Default value	Unit
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
Speed *Speed of the platform that holds the sonar.*	Double	10	Knots

Table 3.20 Parameters in the platform class.

3.16 Sensor class

The sensor class contains all the relevant information about the sonar. The parameters in the sensor class are listed in Table 3.21. There are no access functions in to the sensor class.

Parameter	Type	Default value	Unit
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
PassiveBandWidth *Band width of the passive sonar.*	Double	100	Seconds

Parameter	Type	Default value	Unit
PassiveFrequency *Centre frequency of the passive sonar.*	Double	800	Hz
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 broad- or narrowband.* *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
TiltTransmitter *Tilt of the transmitting part of the sonar.*	Double	4	Degrees

Table 3.21 Parameters in the sensor class.

3.17 Pulse class

All the information about the pulse is gathered in the pulse class. All the access parameters in the pulse class are listed in Table 3.22 below. The pulse class does not have any access functions.

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 3.22 Parameters in the pulse class.

4 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.

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

Table 4.1 Function for initiation of calculation.

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

5 Calculation results

The calculation results can be accessed through parameters or functions found in LybinModelData. The result parameters are listed in Table 5.1. Each parameter represents a complete dataset. The result functions give more flexibility in the way that you can access the calculated results. All the functions delivering calculation results are listed in Table 5.2. If a calculation fails, the returned value properties will be NULL.

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*]	%
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 5.1 Parameters containing 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[2] in the range corresponding to pIndex as an array. Each family has the following order of parameters:* *[,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* *[,1] Intensity loss (double)* *[,2] Mean arrival time – first arrival time in seconds (double)* *[,3] Arrival time standard deviation in seconds (double)* *[,4] Phase identifier[1] (double)* *[,5] First arrival in seconds (double)*	Object[x,6]
<u>GetImpulseResponseFamily(int pIndex, int pFamilieIndex, out string pFamiliName, out double pIntensity, out double pMeanArrivalTime, out double pStandardDeviation, out double pPhase, out double pFirstArrival)</u> *Returns the calculated ray family identifier, intensity, mean arrival time, arrival time standard deviation, phase and first arrival from one single ray family.* *<u>pIndex</u> represents the corresponding range.*	String, Double, Double, Double, Double, Double

[1] 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
pFamilieIndex is the running number of the family at the specified range, resulting from the ray tracing calculation. There is no direct connection between pFamilieIndex and pFamiliName. pFamiliName 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*[1]. *pFirstArrival is the time of the first arrival in seconds.*	
GetImpulseResponseNumFamilies(int pIndex) *Returns the number of different ray families*[2] *at the range corresponding to pIndex.*	Integer
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 5.3.	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 5.3.	Format depends on type of returned object. See Table 5.1.
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 5.3.	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:* *[,0] Initial ray angle (degrees)*	Double[x,4]

[2] 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.

Function	Result format
[,1] Range (meters) *[,2] Depth (meters)* *[,3] Travel time (seconds)*	
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:* *[,0] Initial ray angle (radians)* *[,1] Range (meters)* *[,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 5.2   Functions delivering calculation results

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

Table 5.3 Available values of resultCat with description.

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

```
LybinCom.LybinModelComBinClass Lybin = new
LybinCom.LybinModelComBinClass();

public double[] GetMaskingLevel()
{
    // Initiate the masking level array
    int NumberOfValues = Lybin.RangeCells;
    double[] MaskingLevelValues = new double[NumberOfValues];

    Object Objekt;
    Lybin.GetResultsBin(10, out Objekt);
    MaskingLevelValues = (double[])Objekt;

    return MaskingLevelValues;
}

public double GetNoise()
{
    double noise;
    Object Objekt;

    Lybin.GetResultsBin(8, out Objekt);
    noise = (double)Objekt;

    return noise;
}

public string GetUsedParameters()
{
    string parameters;
    Lybin.GetResultModelData(out parameters);

    return parameters;
}
```

The results from the impulse response calculations can only be accessed through the binary interface. The following gives an example of how to use the impulse response functions. The example is written in C#. The passive frequency and the Rayleigh bottom loss are also set in this example because these parameters are to be used in the calculation of the impulse response.

```
Lybin = new LybinCom.LybinModelComBinClass();

Lybin.PassiveFrequency = 500;

// Rayleigh parameters
double[,] rbl = new double[1, 3];
rbl[0, 0] = 0.92; // bottomAttenuation;
rbl[0, 1] = 1717; // bottomSoundSpeed;
rbl[0, 2] = 1.81; // densityRatio;
Lybin.RayleighBottomLoss = rbl;
Lybin.UseRayleighBottomLoss = true;

// Initiate impulse response
Lybin.ImpulseResponseCalculation = true;
Lybin.ImpulseResponseDepth = 100;

Lybin.DoCalculation();

// Number of ranges calculated
int ranges = Lybin.ImpulseResponseNumRanges;

// Number of families at range number 4
int numFamilies = Lybin.GetImpulseResponseNumFamilies(4);

// Get values from family 5 at range 4:
string familyName;
double intensity;
double meanArrivalTime;
double arrivalTimeStandardDeviation;
double phase;
double firstArrival;

Lybin.GetImpulseResponseFamily(4, 5, out familyName, out intensity,
out meanArrivalTime, out arrivalTimeStandardDeviation, out phase,
out firstArrival);

// Get all families at range 4
object families = Lybin.GetImpulseResponseFamiliesAsArray(4);
```

The results from the travel time calculations can only be accessed through the binary interface. The following gives an example of how to use the travel time functions. The example is written in C#:

```
Lybin = new LybinCom.LybinModelComBinClass();
Lybin.TravelTimeCalculation = true;
Lybin.TravelTimeAngleRes = 0.5;
Lybin.DoCalculation();

Object obj;
Object obj2;
// Number of rays calculated
int pathCount = Lybin.TravelTimePathCount;
//Length of middle ray
int travelLength = Lybin.GetTravelTimePathLength(pathCount / 2);
double[,] values = new double[travelLength, 4];
obj = Lybin.GetTravelTimePathAsDoubleArray(pathCount / 2);
obj2 = Lybin.GetTravelTimePath(pathCount / 2);
if ( obj2 is LybinCom.TravelTimePoint[])
{
     LybinCom.TravelTimePoint[] ttp = (LybinCom.TravelTimePoint[])obj2;
}
values = (double[,])obj;
```

The results from the visual ray trace calculations can only be accessed through the binary interface. The following gives an example of how to use the visual ray trace functions. The example is written in C#:

```
lybin = new LybinCom.LybinModelComBinClass();
lybin.VisualRayTraceCalculation = true;
lybin.VisualSurfaceHits = 6;
lybin.VisualBottomHits = 8;
lybin.VisualNumRays = 66;
lybin.DoCalculation();


Object obj;
// Number of rays calculated
int pathCount = lybin.VisualRayTraceCount;
//Length of middle ray
int travelLength = lybin.GetVisualRayTraceLength(pathCount / 2);
double[,] values = new double[travelLength, 3];
obj = lybin.GetVisualRayTrace(pathCount / 2);
values = (double[,])obj;
```

Appendix A Example code

A.1 C# Windows forms application using LybinCom

```
using System;
using System.Windows.Forms;
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.ReverberationZones = 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 file using LybinCom, basic 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 = 20000;
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;
```

```
% 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 = [0,200;5000,400; 10000,600];
env.bottomProfile = prof;

% Calculate
lb.DoCalculation

% 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;

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

% Parameters for plotting
r = R/R_cells * [.5:(R_cells-.5)];
z = Z/Z_cells * [.5:(Z_cells-.5)];

% Plot data
figure(1)
contourf(r/1000,z,data.trl.forward, 30,'linestyle', 'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
ylabel('Depth [m]')
title('Transmission loss')
colorbar;
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');
hold off

figure(2)
contourf(r/1000,z,data.sig, 30,'linestyle', 'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
```

```
ylabel('Depth [m]')
title('Signal excess')
colorbar;
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');
hold off

figure(3)
contourf(r/1000,z,data.pod, 30,'linestyle', 'none')
set(gca, 'ydir', 'reverse')
xlabel('Range [km]')
ylabel('Depth [m]')
title('Propability of detection')
colorbar;
hold on
fill([prof(:,1);0;0]/1000, [prof(:,2);Z;Z], 'k');
hold off

figure(4)
plot(r/1000, data.rev.Tot_rev)
hold on
plot(r/1000, data.rev.Bot_rev, 'r')
plot(r/1000, data.rev.Surf_rev, 'g')
plot(r/1000, data.rev.Vol_rev, 'c')
title('Reverberation')
legend('Total', 'Bottom', 'Surface', 'Volume')
xlabel('Range [km]')
ylabel('Rev [dB]')
grid on
hold off
```

A.3 Matlab file using LybinCom, impulse response example

```
% Use LybinCom to calculate impulse response %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

% Interfaces
env = lb.invoke('IEnvironment'); % Environment
mod = lb.invoke('IModelData'); % Model
sensor = lb.invoke('ISensor'); % Sonar

% Frequency
sensor.PassiveFrequency = 500;

% Rayleigh bottom loss
bottomAttenuation = 0.92;
bottomSoundSpeed = 1717;
densityRatio = 1.81;
env.RayleighBottomLoss = [bottomAttenuation, bottomSoundSpeed,
densityRatio];
mod.UseRayleighBottomLoss = true;

% Initiate impulse response
```

```
mod.ImpulseResponseCalculation = true;
mod.ImpulseResponseDepth = 50;

% Start calculation
lb.DoCalculation;

% Number of ranges calculated
numRanges = lb.ImpulseResponseNumRanges;

% Number of ramilies at range number 30
numFamilies = lb.GetImpulseResponseNumFamilies(30);

% Get all families at range 30
families = lb.GetImpulseResponseFamiliesAsArray(30);

%Get one specific family at range 30 and family number 2
[Success, FamilyName, Intensity, MeanArrivalTime, StandardDeviation,
Phase, FirstArrival]...
    = lb.GetImpulseResponseFamily(30,2);

% Used input parameters
modelData = lb.ModelData;

% Release interfaces
env.release; % Environment
mod.release; % Model
sensor.release; % Sonar
lb.release; % Component
```

A.4 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(&typeOfRevNoiseCalculation);

   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;
}
```

References

[1]	S Mjølsnes (2000): *LYBIN SGP-180(C) – Model Description*, The Royal Norwegian Navy Materiel Command, Bergen
[2]	E Dombestein and K T Hjelmervik (2004): *Analysis of the NAT III experiments – Modelling assumtions*, Norwegian Defence Research Institute, FFI/RAPPORT 2004/01083
[3]	Teleplan GLOBE, http://www.teleplanglobe.com/index.php?page=maria
[4]	A Gjersøe and F Hermansen (2008): *Simson Fennikel – Design dokument for Maria add-in for utvelgelse og visning av data på kart*, Norwegian Defence Research Institute, FFI/RAPPORT 2008/02182
[5]	M Bosseng and A Gjersøe (2008): *Simson Fennikel – User Manual*, Norwegian Defence Research Institute, FFI/RAPPORT 2008/02183
[6]	E Dombestein and S Alsterberg (2006): *LYBIN XML grensesnitt versjon 1*, Norwegian Defence Research Institute, FFI/RAPPORT 2006/00266
[7]	E Dombestein (2009): LYBIN 5.0 – interface description, Norwegian Defence Research Institute, FFI/RAPPORT 2009/00188
[8]	E Dombestein, A Gjersøe and M Bosseng (2009): LybinCom 6.0 – description of the binary interface, Norwegian Defence Research Institute, FFI/RAPPORT 2009/02267
[9]	E Dombestein, A Gjersøe, K T Hjelmervik and M Kloster (2011): LYBIN 6.0 – user manual, Norwegian Defence Research Institute, FFI/RAPPORT 2011/00205
[10]	A Ishimaru (1978): Wave propagation and scattering in random media, Random press

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

Definitions

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