CNOSSOS-EU

Develop and implement harmonized noise assessment methods.

Task 2: Propagation software modules

User's and programmer's guide

Dirk Van Maercke
Centre Scientifique et Technique du Bâtiment
24 rue Joseph Fourier
38400 Saint Martin d'Hères
FRANCE
dirk.van-maercke@cstb.fr
www.cstb.fr

This document has been prepared on behalf of the European Commission under Service contract number: 070307/2012/633745/ENV.C3.

This document has been produced on behalf of the European Commission under Service Contract N° 070307/2012/633745/ENV.C3.

Table of Contents

1 Introduction

1.1 Implementation of CNOSSOS-EU

As part of the CNOSSOS-EU project, Extrium has been appointed by the European Commission to undertake the development and the implementation of parts of the future harmonized noise assessment methods as foreseen by the European Noise Directive COM 2000/49/EC.

The contract includes the development and implementation of input databases and software modules for harmonised noise assessment methods, based on the outcome and decision made at earlier stages of the CNOSSOS-EU project, which the European Commission intends to make available under open source licensing terms for use by developers of noise mapping software.

Extrium has sub-contracted part of the deliverables and services foreseen under the main contract to CSTB, including the implementation of three software modules for the prediction of the propagation of noise along well-defined propagation paths.

1.2 Objectives

During phase A of the CNOSSOS-EU project there has been much discussion within Working Group 5 regarding the appropriate choice of the propagation model to be included within the harmonised methodology. The Draft JRC Reference Report Version 2d of 28th May 2010 includes a description of the sound propagation model which is based upon, and closely related to, the outcomes of the Harmonoise and Imagine projects. Following this, WG5 undertook a comparative review which included the ISO 9613-2 standard alongside the Harmonoise/Imagine methodology and the French NMPB 2008 methodology. The JRC Reference Report of 5th June 2012 then includes an updated description of the sound propagation model, based upon, and closely related to, the NMPB 2008 methodology.

However this recommendation did not constitute a universal and robust conclusion to the discussions within WG5, partly because the three methodologies were not available in consistent implementations, which compromised the results of comparisons between the three methods For this reason the project sets out to provide software modules for the three point-to-point methodologies, such that each may be tested further as part of phase B of the CNOSSOS-EU process.

Under the provisions of the sub-contract, CSTB has undertaken to implement these three propagation modules based on the technical descriptions as found in:

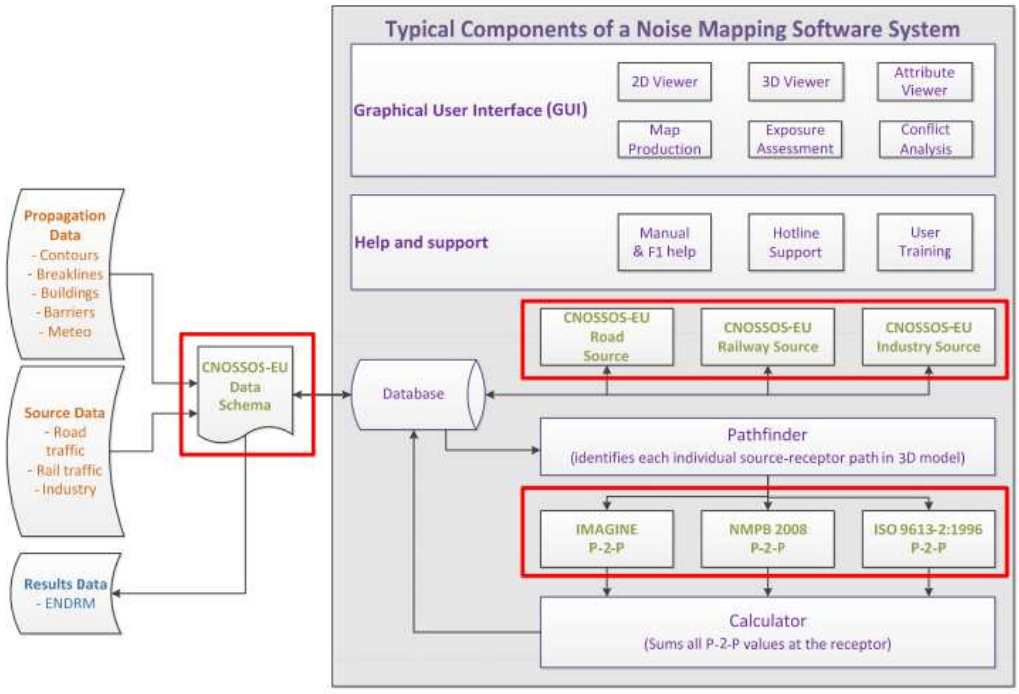
- The international standard ISO 9613-2,
- The JRC Reference Report "Common Noise Assessment Methods in Europe (CNOSSOS-EU)", draft version issued in May 2010 and the associated documentation as delivered by the Harmonoise and Imagine projects.

- The JRC Reference Report "Common Noise Assessment Methods in Europe (CNOSSOS-EU)", final version issued in August 2012, and the related French standard NF S31-133.

Where applicable, additional definition, revisions and clarifications with respect to these documents have been taken from the respective Project Issue Logs maintained for the duration of the project.

The input to the propagation modules contains a geometrical description of the propagation path in 3D coordinates, associated material properties and meteorological conditions influencing sound propagation. The output of each module is the sound level at the receiver for individual propagation paths.

As can be seen in Figure 1 below, the software modules do not constitute a full noise mapping software tool, as they do not include a pathfinder capability capable of identifying the point to point paths within a 2D or 3D model dataset, nor do they include a noise calculator capable of summing the individual point to point paths at a given receptor location.



Similarly the software modules do not include a graphical user interface (GUI) and 2D or 3D model data viewers as commonly found in commercial applications. Rather, the software modules are software components which may be integrated into other 3rd party tools in the future with reference to the present documentation.

In view of the objectives in general and the need for comparison in particular, care has been taken to ensure that all three propagation modules can be run on a single set of input data and to provide detailed outputs enabling the comparison of the three methods at the level of individual components in the noise calculation.

In view of the foreseen use, the propagation modules provide several kinds of interface enabling both interactive testing and integration in 3rd party software tools, including:

- Input/output by means of XML-formatted files,
- Standalone test software for manual testing,
- C-style programming interface,
- C++ programming interface.

1.3 Contents of this document

Section 2 of this document provides information on installing and running the standalone test software.

Section 3 describes in detail how to manual create XML-formatted input files. This information, used in conjunction with the standalone test software allows for the comparison of the three propagation models on user-defined test cases without relying on programming skills or integration of the modules in 3rd party software packages.

Section 4 reports on the testing of the software modules. Testing concerns both the logical conformity of the software with respect to the expected behaviour as described in the documentation and the plausibility of the results with respect to commonly accepted features of outdoor noise propagation. Experimental validation of the results is deemed outside the scope of the current project.

Section 5 describes how to modify and rebuild the software using either the Microsoft Visual Studio C++ or the MingW compiler (a freeware implementation of the GCC compiler for MS Windows systems).

Section 6 illustrates how to integrate the propagation modules into a larger software packages using the shared library and the C-style application programming interfaces.

Section 7 illustrates the advanced use of the libraries exploiting the direct access to the C++ programming interface, e.g. for the purpose of modifying and extending the functionality of the three propagation modules without breaking the compatibility and functionality of the three methods as provided under the current project.

In detail information on some aspects of the methods and their implementation is provided in the annexes.

- Annexe A summarizes the licensing conditions of the different software components.
- Annexe B contains the reference documentation for the XML input file format;
- Annexe C contains the reference documentation for the C-style application programming interface to the shared library containing the three propagation modules;
- Annexe D documents the implementation of basic geometrical operations;

- Annexe E documents the common geometrical operations (including validity checking) on propagation paths which are shared by all three propagation modules;
- Annexe F provides additional details of the implementation of the acoustical calculations insofar that those are lacking or not sufficiently clear in the reference documents;

2 Installing an running the software

2.1 System requirements

This software is intended to run on any PC computer conforming to these minimal requirements:

- Intel© or AMD© 32 bits (x86) or 64 (x64) processor,
- 1 GHz processor speed or better,
- Windows 7 operating system (or more recent if compatible),
- At least 500 Mb free disk space for downloading and uncompressing the software distribution files.

2.2 Download and install the software

Step 1: download the software

Ask Dirk for the download address, indicating the preferred method: FTP server or DropBox. Note that you can download from Dropbox without having a login account.

Step2: unzip the file into a folder of your choice

Note: free zip/unzip software can be found on the web. I recommend "7zip" freeware which is available under GNU LGPL from http://www.7-zip.org/

Note: the download directory contains the following folders:

Cnossos-EU	Building blocks (including source code) for the test application
CnossosPropagation	Source code of the Cnossos Propagation shared libray
Data	Sample input files used for verification of the software
Debug	Output of the build process in debug mode
HarmonoiseP2P	Source code for the Harmonoise point-to-point library
PropagationPath	Source code for the for the calculation methods
Release	Output of the building process in release mode
redist_msvcrt_crt	Redistributable files for Microsoft Visual Studio 2010 CRT library
SimpleXML	Source code for the XML parser library (including the Expat library)
TestCnossosCPP	Test software illustrating the use of the propagation library in C++
TestCnossosDLL	Test software illustrating the use of the shared library in C
TestCnossosEXT	Test software illustrating advanced programming in C++

Step3: open a command window and go to the download folder.

Note: under the Windows OS, a free application like "FileMenu Tools" can be of great help. Navigate to the right directory using Windows Explorer. Then right click and select "Command Line From Here" from the "FileMenu Tools" menu item. The FileMenu freeware can be downloaded from http://www.lopesoft.com/en/filemenutools

Step 4: select the build you want to use.

The executable comes in two flavours: release and debug. The debug version prints out some intermediate results whereas the release version does not. For normal operation, I recommend using the release build.

When called without any command line options, the program prints out a summary of available options:

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\>cd Release
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>TestCnossos

Usage:

  TestCnossos [-w] [-m=<method>] [-i=<input file>] [-c] [-o] [-o=<outputfile>]

  .if -w is specified, the program will halt and wait for some user input
   before exiting, otherwise exit is automatic at the end of the calculations

  .if -c is specified, the program will copy the results to the clipboard in a
   tabular format ready for pasting in spreadsheet applications or text editors

  .method = ISO-9613-2, JRC-draft-2010 or JRC-2012; if not specified, the
   calculation will use the default method as found in the input file.

  .input file = the name of a valid XML file complying to the CNOSSOS-EU
   specifications.

  .output file = the name of the output file. If the file already exists,
   if will be replaced. If no filename is specified, output will be sent to
   the standard output stream for the active process.

   note that the output file will be searched and/or created relative to
   the folder containing the input file. If the file exists, it will be
   replaced or deleted (in case the program reports an error and no results
   can be produced).

   if -o is specified without a file name, the program will automatically
   generate one, based on the name of the input file name.
```

2.3 Running predefined test cases

The ..\data folder contains a set of predefined test cases, mainly used for the validation and the testing of the different software components. These test cases can also be used to check the correct installation of the software on your computer.

Step 5: list the available test cases

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>dir..\data\*.xml
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est 3208-A6F9

 Répertoire de C:\Users\dvm\Projets\CNOSSOS-EU\dev\data

11/12/2013  13:18    <REP>          .
11/12/2013  13:18    <REP>          ..
05/12/2013  15:12             2 436 area source.xml
02/12/2013  14:56             1 565 barrier 1m - 100m.xml
02/12/2013  14:56             1 565 barrier 1m - 200m.xml
02/12/2013  14:56             1 565 barrier 1m - 20m.xml
02/12/2013  14:56             1 565 barrier 1m - 500m.xml
02/12/2013  14:56             1 565 barrier 1m - 50m.xml
29/10/2013  09:25             1 567 barrier 2m - 100m.xml
29/10/2013  09:26             1 567 barrier 2m - 200m.xml
29/10/2013  09:25             1 565 barrier 2m - 20m.xml
29/10/2013  09:27             1 552 barrier 2m - 500m.xml
29/10/2013  09:26             1 565 barrier 2m - 50m.xml
02/12/2013  14:58             1 565 barrier 4m - 100m.xml
02/12/2013  14:58             1 565 barrier 4m - 200m.xml
02/12/2013  14:57             1 565 barrier 4m - 20m.xml
02/12/2013  14:58             1 565 barrier 4m - 500m.xml
02/12/2013  14:58             1 565 barrier 4m - 50m.xml
28/11/2013  09:45               507 CNOSSOS_Road_Output.xml
02/12/2013  14:41             1 624 depressed - 100m.xml
02/12/2013  14:40             1 624 depressed - 10m.xml
02/12/2013  14:42             1 624 depressed - 200m.xml
02/12/2013  14:40             1 624 depressed - 20m.xml
02/12/2013  14:42             1 624 depressed - 500m.xml
02/12/2013  14:41             1 624 depressed - 50m.xml
02/12/2013  14:46             1 717 depressed+barrier - 100m.xml
02/12/2013  14:45             1 717 depressed+barrier - 10m.xml
02/12/2013  14:46             1 717 depressed+barrier - 200m.xml
02/12/2013  14:46             1 717 depressed+barrier - 20m.xml
02/12/2013  14:47             1 717 depressed+barrier - 500m.xml
02/12/2013  14:46             1 717 depressed+barrier - 50m.xml
04/11/2013  17:23             1 624 embankment-1m - 100m.xml
04/11/2013  17:22             1 624 embankment-1m - 10m.xml
04/11/2013  17:23             1 624 embankment-1m - 200m.xml
```

```
04/11/2013  17:22             1 624 embankment-1m - 20m.xml
04/11/2013  17:23             1 624 embankment-1m - 500m.xml
04/11/2013  17:22             1 624 embankment-1m - 50m.xml
25/11/2013  14:36             1 622 embankment-2m - 100m.xml
04/11/2013  17:23             1 624 embankment-2m - 10m.xml
04/11/2013  17:24             1 624 embankment-2m - 200m.xml
04/11/2013  17:24             1 624 embankment-2m - 20m.xml
04/11/2013  17:25             1 622 embankment-2m - 500m.xml
04/11/2013  17:24             1 624 embankment-2m - 50m.xml
02/12/2013  14:38             1 627 embankment-4m - 100m.xml
02/12/2013  14:37             1 627 embankment-4m - 10m.xml
02/12/2013  14:39             1 627 embankment-4m - 200m.xml
02/12/2013  14:38             1 627 embankment-4m - 20m.xml
02/12/2013  14:39             1 627 embankment-4m - 500m.xml
02/12/2013  14:38             1 627 embankment-4m - 50m.xml
29/10/2013  09:23             1 454 flat ground - 100m.xml
02/12/2013  14:31             1 452 flat ground - 10m.xml
29/10/2013  09:23             1 454 flat ground - 200m.xml
18/11/2013  09:44             1 452 flat ground - 20m.xml
29/10/2013  09:23             1 454 flat ground - 500m.xml
29/10/2013  09:22             1 452 flat ground - 50m.xml
05/12/2013  10:27             1 452 high-source - 100m.xml
05/12/2013  10:20             1 452 high-source - 10m.xml
05/12/2013  10:24             1 452 high-source - 200m.xml
05/12/2013  10:27             1 452 high-source - 20m.xml
05/12/2013  10:27             1 452 high-source - 500m.xml
05/12/2013  10:22             1 452 high-source - 50m.xml
05/11/2013  15:01             2 710 lateral diffraction not-convex.xml
05/11/2013  15:02             2 710 lateral diffraction too-high.xml
05/11/2013  14:38             2 710 lateral diffraction.xml
05/12/2013  15:19             2 358 line source.xml
05/12/2013  15:19             2 328 point source + road traffic.xml
05/12/2013  15:19             2 410 point source.xml
06/12/2013  16:42             1 600 rec_to_source.xml
05/12/2013  14:32             2 751 reflection-high-comp.xml
29/10/2013  12:19             2 753 reflection-high.xml
05/12/2013  14:33             2 751 reflection-low-comp.xml
29/10/2013  10:42             2 753 reflection-low.xml
05/12/2013  14:33             2 751 reflection-mid-comp.xml
29/10/2013  13:01             2 753 reflection-mid.xml
05/12/2013  14:56             2 397 road traffic + segment + height.xml
05/12/2013  14:55             2 335 road traffic + segment.xml
09/12/2013  13:37             2 505 simplify terrain off.xml
09/12/2013  13:37             2 504 simplify terrain on.xml
06/12/2013  17:26             1 599 source_to_rec.xml
05/12/2013  11:05             1 772 trench - 1refl - 100m.xml
05/12/2013  11:03             1 772 trench - 1refl - 10m.xml
05/12/2013  11:05             1 772 trench - 1refl - 200m.xml
05/12/2013  11:04             1 772 trench - 1refl - 20m.xml
```

```
05/12/2013  11:05             1 772 trench - 1refl - 500m.xml
05/12/2013  11:04             1 772 trench - 1refl - 50m.xml
05/12/2013  12:24             1 973 trench - 2refl - 100m.xml
05/12/2013  12:22             1 973 trench - 2refl - 10m.xml
05/12/2013  12:24             1 973 trench - 2refl - 200m.xml
05/12/2013  12:23             1 973 trench - 2refl - 20m.xml
05/12/2013  12:24             1 973 trench - 2refl - 500m.xml
05/12/2013  12:23             1 973 trench - 2refl - 50m.xml
05/12/2013  12:30             2 174 trench - 3refl - 100m.xml
05/12/2013  12:25             2 174 trench - 3refl - 10m.xml
05/12/2013  12:30             2 174 trench - 3refl - 200m.xml
05/12/2013  12:29             2 174 trench - 3refl - 20m.xml
05/12/2013  12:30             2 174 trench - 3refl - 500m.xml
05/12/2013  12:29             2 174 trench - 3refl - 50m.xml
05/12/2013  10:59             1 571 trench - direct - 100m.xml
05/12/2013  10:58             1 571 trench - direct - 10m.xml
05/12/2013  10:59             1 571 trench - direct - 200m.xml
05/12/2013  10:58             1 571 trench - direct - 20m.xml
05/12/2013  11:00             1 571 trench - direct - 500m.xml
05/12/2013  10:59             1 571 trench - direct - 50m.xml
02/12/2013  16:35             1 720 valley - 100m.xml
02/12/2013  16:36             1 842 valley - 200m.xml
05/12/2013  10:48             1 431 valley - 20m.xml
02/12/2013  16:37             1 964 valley - 500m.xml
02/12/2013  16:34             1 597 valley - 50m.xml
             119 fichier(s)        1 225 404 octets
               2 Rép(s)  170 021 474 304 octets libres
```

Step 6: run the program using one of the predefined test cases:

The input file is specified using the "-i" command line option, as shown in the example below: the same test cases is now run through the JRC-2012 calculation schemes instead of the ISO-9613-2.

Note: if the "-m" option is not specified, the test software will use the default calculation method encoded in the input file (see next section).

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>TestCnossos -w
-i="..\data\barrier 2m - 200m.xml"
Parse file: ..\data\barrier 2m - 200m.xml
Calculate path
.Method:  ISO-9613-2:1996
.Version: 1.001
.Meteo:   ISO-9613-2, C0=3.0 dB
Unfolded propagation path
-------------------------------------------------------------------------
ID |       D |      Z |      H |  Z_ray |    G | Mode
-------------------------------------------------------------------------
00 |    0.00 |   0.00 |   0.50 |   0.50 | 0.00 | SRC
```

```
01 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
02 |   10.00 |   2.00 |   0.00 |   2.00 | 0.00 | DIF
03 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
04 |  200.00 |   0.00 |   2.50 |   2.50 | 1.00 | REC
--------------------------------------------------------------------------
.1000 calls to P2P calculation in 11.608510ms
Noise levels
--------------------------------------------------------------------------
Freq(Hz) |    63 |   125 |   250 |   500 |  1000 |  2000 |  4000 |  8000
--------------------------------------------------------------------------
Lw       |  80.0 |  90.0 |  95.0 | 100.0 | 100.0 | 100.0 |  95.0 |  90.0
dB(A)    | -25.2 | -15.6 |  -8.4 |  -3.1 |   0.0 |   1.2 |   0.9 |  -2.4
deltaLw  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttGeo   | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0
AttAtm   |  -0.0 |  -0.1 |  -0.2 |  -0.5 |  -0.8 |  -1.8 |  -5.3 | -19.0
AttRef   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttDif   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttSize  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
Att,F    |  -5.0 |  -5.3 |  -5.8 |  -6.6 |  -7.8 |  -9.6 | -11.8 | -14.4
Att,H    |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf
Lp,F     |  -7.3 |  12.0 |  23.6 |  32.9 |  34.4 |  32.9 |  21.8 |  -2.8
Lp,H     |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf |  -inf
Leq      |  -9.8 |   9.5 |  21.1 |  30.3 |  31.8 |  30.3 |  19.2 |  -5.3
--------------------------------------------------------------------------
Lp,F     |  38.4 dB(A)
Lp,H     |  -inf dB(A)
Leq      |  35.9 dB(A)
--------------------------------------------------------------------------
Type any key to quit this program
```

Step 7: select the calculation method

Although the calculation method is specified as part of the input file, any test case can be run using an alternative the calculation method. Alternative calculation methods are selected using the "-m" command line argument. Valid methods are "ISO-9613-2", "JRC-2012" and "JRC-draft-2010".

Note that all three calculation methods use strictly identical input files although some method may ignore some input values. For most input parameters, the software will use predefined equivalences for adapting the common "external" identifiers into equivalent "internal" values used by the different calculation schemes.

In the next section, we'll learn how these predefined equivalences can be modified by the end-user.

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>TestCnossos -w
-i="..\data\barrier 2m - 200m.xml" -m="JRC-2012"
Parse file: ..\data\barrier 2m - 200m.xml
Calculate path
```

```
.Method:  JRC_final_2012
.Version: 1.001
.Meteo:   JRC-2012, pFav=50.0%
Unfolded propagation path
-------------------------------------------------------------------------
ID |       D |      Z |      H |  Z_ray |    G | Mode
-------------------------------------------------------------------------
00 |    0.00 |   0.00 |   0.50 |   0.50 | 0.00 | SRC
01 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
02 |   10.00 |   2.00 |   0.00 |   2.00 | 0.00 | DIF
03 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
04 |  200.00 |   0.00 |   2.50 |   2.50 | 1.00 | REC
-------------------------------------------------------------------------
.1000 calls to P2P calculation in 39.940342ms
Noise levels
-------------------------------------------------------------------------
Freq(Hz) |    63 |   125 |   250 |   500 |  1000 |  2000 |  4000 |  8000
-------------------------------------------------------------------------
Lw       |  80.0 |  90.0 |  95.0 | 100.0 | 100.0 | 100.0 |  95.0 |  90.0
dB(A)    | -25.2 | -15.6 |  -8.4 |  -3.1 |   0.0 |   1.2 |   0.9 |  -2.4
deltaLw  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttGeo   | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0
AttAtm   |  -0.0 |  -0.1 |  -0.2 |  -0.5 |  -0.8 |  -1.8 |  -5.3 | -19.0
AttRef   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttDif   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttSize  |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0
Att,F    |  -3.0 |  -3.8 |  -8.0 | -10.4 |  -9.1 | -11.7 | -14.4 | -17.3
Att,H    |  -3.1 |  -4.1 |  -7.4 | -17.1 | -11.7 | -12.3 | -15.1 | -18.0
Lp,F     |  -5.2 |  13.5 |  21.4 |  29.0 |  33.0 |  30.8 |  19.1 |  -5.7
Lp,H     |  -5.4 |  13.2 |  22.0 |  22.3 |  30.5 |  30.1 |  18.4 |  -6.4
Leq      |  -5.3 |  13.4 |  21.7 |  26.8 |  31.9 |  30.4 |  18.8 |  -6.1
-------------------------------------------------------------------------
Lp,F     |  36.3 dB(A)
Lp,H     |  34.1 dB(A)
Leq      |  35.3 dB(A)
-------------------------------------------------------------------------
Type any key to quit this program
```

Note that the order of the command options is not relevant.

2.4 Interpreting the results

For ease of use, the test program prints out the results of the acoustical calculations on the standard output device, i.e. the console window from which the command line was entered by the user.

Note that the table shows attenuations as "level differences", so that higher values correspond to higher levels, no matter if the value represents an absolute or a relative noise level. For attenuation terms, positive values represent an increase in noise level whereas negative values correspond to noise reduction.

The results are presented in tabular format and correspond to the individual components entering the evaluation of the final noise level.

Symbol	Meaning
Lw	Sound power of the elementary source
dB(A)	A-weighting (set to zero if Lw is already A-weighted)
deltaLw	Sound power conversion (free field versus hemispherical radiation)
AttGeo	Geometrical spread
AttAtm	Atmospheric absorption according to ISO 9613-1
AttRef	Attenuation due to absorbing materials on reflecting surfaces
AttDif	Attenuation due to lateral diffraction by vertical edges
AttSize	Correction for finite size of vertical obstacles
Att,F	Excess attenuation under favourable propagation conditions
Att,H	Excess attenuation under homogeneous propagation conditions
Lp,F	Noise levels under favourable propagation conditions
Lp,H	Noise levels under homogeneous propagation conditions
Leq	Long-time averaged noise levels

Note that the several components are common to all three calculations methods (e.g. atmospheric absorption or geometrical spread) whereas other components have specific implementations in each method (e.g. excess attenuation and corrections for finite size of vertical obstacles).

Component	Common	ISO-9613-2	JRC-2012	JRC-draft-2010
Lw	X			
dB(A)	X			
deltaLw	X			
AttGeo	X			
AttAtm	X			
AttRef	X			
AttDif	N.A.	X	X	N.A.
AttSize	N.A.	X	X	X
Att,F	N.A.	X	X	X
Att,H	N.A.	N.A.	X	X

Lp,F	X			
Lp,H	X			
Leq	X			

In case a component is not applicable or not implemented in one of the methods, the calculation returns either an infinite attenuation or returns an error conditions. In both cases, this results in a noise level equal to minus infinity.

Note that internally, there is no special representation of "undefined" values for noise levels or attenuations. The floating point processor is programmed so that it correctly handles the IEEE extended range of real numbers, including mathematical operations on positive or negative infinity insofar that these operations are defined (e.g. log(0) correctly returns minus infinity). This implies that all operations on spectral results (e.g. summation of global dB(A) levels or meteorological weighting) work correctly even in case the spectrum contains one or more missing values.

Alternatively, the output of the program can be redirected into a file saved on the hard disk:

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>TestCnossos
-i="..\data\barrier 2m - 200m.xml" -m="JRC-2012" >>  test.tmp

C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>type test.tmp
Parse file: ..\data\barrier 2m - 200m.xml
Calculate path
.Method:  JRC_final_2012
.Version: 1.001
.Meteo:   JRC-2012, pFav=50.0%
Unfolded propagation path
------------------------------------------------------------------------
ID |       D |      Z |      H |  Z_ray |    G | Mode
------------------------------------------------------------------------
00 |    0.00 |   0.00 |   0.50 |   0.50 | 0.00 | SRC
01 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
02 |   10.00 |   2.00 |   0.00 |   2.00 | 0.00 | DIF
03 |   10.00 |   0.00 |   0.00 |   2.00 | 0.00 |
04 |  200.00 |   0.00 |   2.50 |   2.50 | 1.00 | REC
------------------------------------------------------------------------
.1000 calls to P2P calculation in 42.065679ms
Noise levels
------------------------------------------------------------------------
Freq(Hz) |    63 |   125 |   250 |   500 |  1000 |  2000 |  4000 |  8000
------------------------------------------------------------------------
Lw       |  80.0 |  90.0 |  95.0 | 100.0 | 100.0 | 100.0 |  95.0 |  90.0
dB(A)    | -25.2 | -15.6 |  -8.4 |  -3.1 |   0.0 |   1.2 |   0.9 |  -2.4
deltaLw  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttGeo   | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0 | -57.0
AttAtm   |  -0.0 |  -0.1 |  -0.2 |  -0.5 |  -0.8 |  -1.8 |  -5.3 | -19.0
```

```
AttRef    |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttDif    |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttSize   |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0 |  -0.0
Att,F     |  -3.0 |  -3.8 |  -8.0 | -10.4 |  -9.1 | -11.7 | -14.4 | -17.3
Att,H     |  -3.1 |  -4.1 |  -7.4 | -17.1 | -11.7 | -12.3 | -15.1 | -18.0
Lp,F      |  -5.2 |  13.5 |  21.4 |  29.0 |  33.0 |  30.8 |  19.1 |  -5.7
Lp,H      |  -5.4 |  13.2 |  22.0 |  22.3 |  30.5 |  30.1 |  18.4 |  -6.4
Leq       |  -5.3 |  13.4 |  21.7 |  26.8 |  31.9 |  30.4 |  18.8 |  -6.1
-------------------------------------------------------------------------
Lp,F      |  36.3 dB(A)
Lp,H      |  34.1 dB(A)
Leq       |  35.3 dB(A)
-------------------------------------------------------------------------
```

The output can also be send to an XML file for further processing by other applications. The name of the XML file is specified by means of the "-o=<filename>" or "-o" command line options. Note that, when using the "-o" option, the program will generate a default name for the output file, starting from the name of the input file and the folder it is located in.

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>TestCnossos -m="JRC-2012" -o
-i="..\data\barrier 2m 200m.xml" >> test.tmp
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release> cd ..\data
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release>type "barrier 2m - 200m.out.xml"
<?xml version="1.0" encoding="UTF-8" ?>
<CNOSSOS-EU>
  <pathResult>
    <LeqA> 35.3 </LeqA>
    <Leq>
        -5.3    13.4    21.7    26.8    31.9    30.4    18.8    -6.1
    </Leq>
    <details>
      <Lw>
          80.0    90.0    95.0   100.0   100.0   100.0    95.0    90.0
      </Lw>
      <dBA>
         -25.2   -15.6    -8.4    -3.1     0.0     1.2     0.9    -2.4
      </dBA>
      <deltaLw>
           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
      </deltaLw>
      <attGeo>
         -57.0   -57.0   -57.0   -57.0   -57.0   -57.0   -57.0   -57.0
      </attGeo>
      <attAir>
          -0.0    -0.1    -0.2    -0.5    -0.8    -1.8    -5.3   -19.0
      </attAir>
      <attRef>
           0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
      </attRef>
```

```
        <attDif>
             0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
        </attDif>
        <attSize>
            -0.0    -0.0    -0.0    -0.0    -0.0    -0.0    -0.0    -0.0
        </attSize>
        <attF>
            -3.0    -3.8    -8.0   -10.4    -9.1   -11.7   -14.4   -17.3
        </attF>
        <attH>
            -3.1    -4.1    -7.4   -17.1   -11.7   -12.3   -15.1   -18.0
        </attH>
        <LpF>
            -5.2    13.5    21.4    29.0    33.0    30.8    19.1    -5.7
        </LpF>
        <LpH>
            -5.4    13.2    22.0    22.3    30.5    30.1    18.4    -6.4
        </LpH>
        <LpFA> 36.3 </LpFA>
        <LpHA> 34.1 </LpHA>
      </details>
    </pathResult>
  </CNOSSOS-EU>
```

Finally, if the "-c" option is specified as part of the command line, the table of results will be copied directly to the clipboard so that it can be pasted in a text processing program or in a spread sheet application. The tabular text format uses <TAB> characters for separating columns and the <CR><LF> combination to separate lines. Most popular spread sheet applications (e.g. Microsoft Excel or OpenOffice Calc) automatically recognize this format.

2.5 Create new test cases

Step 8: modify or create your own test cases

The next section provides a step-by-step description of the input file format. Note that it is generally easier to start from an existing test case and to modify it according to your needs than to start from scratch.

Note: on Windows systems, XML files can be opened by double-clicking them. By default this will open a "Internet Explorer" window showing the contents of the file in a syntax-sensitive way but this application will not allow you to modify the file. Alternatively you may open the file using the NotePad.exe software (which is also part of the Windows OS), but it offers very basic editing functionality. A good freeware for editing XML files (and many other well-known text formats) is Notepad++, which can be found here: http://www.notepad-plus-plus.org/.

Because the XML format has very strict rules, it is not unusual that a hand-written file is rejected by the application. Appropriate error messages may help locating and correcting the syntactic and semantic errors.

2.6 Error messages

The TestCnossos software writes messages to the program's standard output device (i.e. the console window from which the command line was entered or the redirected output file). The format and contents of the messages depend on the context in which the error occurred.

- Level 1: the input file does not exist, could not be accessed, or you don't have sufficient authorisation to access the input file.

```
Parse file: ../data/test5.xml
ERROR: file not found
No results available...
```

- Level 2: the input file does not conform to valid XML syntax. For the precise meaning of error codes, see the documentation of the EXPAT library. Most commonly, the name of some tag is misspelled (tags are case-sensitive), an open tag is not balanced by a closing one, or a slash character "/" is missing in an empty tag (e.g. writing <mat id="A" > will produce this kind of errors because there is no matching end tag). The line and column number generally help to locate the error.

```
Parse file: ../data/test.xml
ERROR: syntax error while parsing file
.internal error code = XML_ERROR_TAG_MISMATCH
.at position (line 66, column 4)
No results available...
```

- Level 3: the input file contains valid XML but does not conform to the Cnossos-EU semantic specification, e.g. some mandatory tag is missing or a tag is present that is not allowed within the current context.

```
Parse file: ../data/test.xml
ERROR: missing tag
.expecting tag <select>
.while parsing tag <options>
.from parent <method>
.from parent <CNOSSOS-EU>
No results available...
```

- Level 4: the value encoded in the tag is not conforming to the Cnossos-EU semantic specifications, i.e. a number is outside the range of allowed values, an identified does not reference a known object, a list of values does not have the right length

```
Parse file: ../data/test.xml
```

```
ERROR: too many values
.while parsing tag <alpha>
.from parent <mat id="UserDefined">
.from parent <materials >
.from parent <CNOSSOS-EU >
.in file <../data/test.xml>
```

- Level 5: The calculation method failed to produce a result for this path because at least one of the input parameters did not conform to the restrictions of the specified method. E.g. in the ISO 9613-2 methods, reflected paths are accepted under the condition that the specular reflection point falls within the vertical limits (lower and upper bound) of the reflecting obstacle whereas this condition is not required in the two other methods.

```
Parse file: ../data/reflection-high.xml
Calculate path
.Method:  ISO-9613-2:1996
.Version: 1.001
.fixed mandatory option: CheckHeightLowerBound
.fixed mandatory option: CheckHeightUpperBound
.using default meteorological model as defined in ISO-9613-2
ERROR: Reflection/diffraction above upper bound at position 2
No results available...
```

2.7 Missing Microsoft Visual Studio Run Time libraries

In order to run, the TestCnossos.exe needs the following dynamic loadable libraries:

```
Process ID: 13024
Using dynamic libraries
C:\Windows\syswow64\ntdll.dll
C:\Windows\syswow64\kernel32.dll
C:\Windows\syswow64\KERNELBASE.dll
C:\Windows\syswow64\PSAPI.DLL
C:\Windows\system32\MSVCR100.dll
C:\Windows\system32\MSVCP100.dll
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release\HarmonoiseP2P.dll
C:\Users\dvm\Projets\CNOSSOS-EU\dev\Release\TestCnossos.exe
```

The files ntdll.dll, kernel32.dll, kernelbase.dll and psapi.dll are part of the Windows operating system and should be found on any computer running Windows XP or later. Note that the name and the location may change from one version to another, i.e. whether the OS is running in 32 or 64 bits native mode.

The file HarmonoiseP2P.dll is delivered and installed as part of the Cnossos-EU Propagation software.

The files MSVCR100.dll and MSVCP100.dll implement the C and C++ run time library and may or may not be present on your computer. These files are installed as part of the Microsoft Visual Studio 2010 development software and developers may distribute them free of charge together with their own products. It is not obvious however if this right to distribute can be transferred to third parties.

If these files are not present on your computer, the Cnossos test software will not run and the operating system will display an error message. In this case, you can use one of the following alternatives to install these files manually:

1) Copy the files from the .\redist_msvcrt_crt folder into the Windows\System32 folder.

2) Copy the files from the .\redist_msvcrt_crt folder into the .\release and .\debug folders.

3) Download and install the Microsoft Visual C++ 2010 Redistributable Package (x86) package which can be downloaded, free of charge (but subject to the acceptance of Microsoft's End-User Licence Agreement) at the following address:

   http://www.microsoft.com/en-us/download/details.aspx?id=8328

3 Input file specifications

3.1 Coding the geometry of the propagation path

The propagation path is a (potentially broken) vertical plane connecting the source to the receiver (or vice-versa). This path may be « broken » due to reflections from or diffractions around vertical obstacles.

The propagation path is fully described by the boundary of the propagation medium, i.e. the interface between the air and the solids (terrain or man-made obstacle) beneath it. The boundary, as taken into account in the acoustical calculations, is the intersection of the vertical propagation plane with the solids.

The boundary can therefore be described by means of a single, connected, polygon line in 3D. The polygon line is made of a set of connected line segments connecting an ordered set of control points. The <path> and <cp> tags are used for coding the sequence of control points. As a minimum, each control point is defined by its position on the ground:

```
<path>
   <cp>
      <pos>
          <x> 10.00 </x>
          <y> 20.00 </y>
          <z> 15.50 </z>
      </pos>
    <cp>
    ...
   <cp>
      <pos>
          <x> 110.00 </x>
          <y> -15.00 </y>
          <z>  17.75 </z>
      </pos>
    <cp>
</path>
```

A valid propagation path must contain at least two control points.

Note:

> The fact that a path definition conforms to the syntax above does not means it represents a valid input data set for the calculation modules; validation of the data as a valid propagation path is described in section…

Optional values:

> If any of the <x>, <y> or <z> tags is missing, the value is assumed to the equal to the one specified for the previous control point. If a value is missing for the first control point is assumed to be equal to zero. Note that this rule makes it easy to construct 2D (or so-called unfolded) propagation paths directly, i.e. by ignoring all <y> values they will be set

to zero and the path will automatically lay in a single vertical plane corresponding to the plane y = 0.

3.2 Coding material properties

Each control point (except the initial one) defines a segment of the boundary polygon. Each segment must be given some acoustical property (from the physical point of view it must be assigned some acoustical properties like e.g. an impedance value).

Material properties are encoded by adding a <mat> tag to the endpoint of the segment.

Materials are encoded using reference identifiers. E.g.

```
<cp>
   <pos>
      <x> 10.00 </x>
      <y> 20.00 </y>
      <z> 15.50 </z>
   </pos>
   <mat id="F" />
<cp>
```

Note:

Materials are given by means of "identifiers". Acoustical properties associated with materials are specific to the calculation method (e.g. JRC-2012 and ISO-9613-2 define ground in terms of G values, whereas the Harmonoise model specifies boundary properties by means of complex impedance values). For each identifier and for each method, appropriate acoustical properties are given in separate tables.

There is a predefined set of identifiers, e.g. as in JRC-2012, p.86. The end-user may extend this table.

Whenever, there is a missing property, the software automatically evaluates a generic value for it based on available information. This includes conversion of G values into equivalent flow resistivity (by means of an empirical formula), evaluation of impedances based on flow resistivity (using the Delany-Bazley one-parameter model) and (analytical) conversion of impedance values into Sabine (diffuse field) absorption coefficients.

Optional values:

If a <mat> tag is missing, its value will be taken equal to the one specified for the previous segment.

It is possible to specify a <mat> property for the first control point. This value will than act as the default value for the next segments until another <mat> is specified.

If the <mat> tag is missing for the first control point, it will be assumed equal to "H", i.e. a perfectly hard surface.

There is a list of predefined identifiers, hard-coded in the software. This list consists of 8 ground types and 5 material classes for vertical walls. This list can be amended and/or extended by the user (see section 3.10).

Identifier	Description	G-value	Sigma in kPa.s/m²
A	Very soft (snow or moss-like)	1.0	12.5
B	Soft forest floor (short, dense heather-like or thick moss)	1.0	31.5
C	Uncompacted, loose ground (turf, grass, loose soil)	1.0	80
D	Normal uncompacted ground (forest floors, pasture field)	1.0	200
E	Compacted field and gravel (compacted lawns, park area)	0.7	500
F	Compacted dense ground (gravel road, parking lot, ISO 10844)	0.3	2000
G	Hard surfaces (most normal asphalt, concrete)	0.0	20000
H	Very hard and dense surfaces (dense asphalt, concrete, water)	0.0	200000

Identifier	Absorption index according to EN 1793-1	G-value	Sigma in kPa.s/m²
A0	Not measured	0.0	20000
A1	$0 < DL\alpha < 4$ dB	0.3	2000
A2	$4 \leq DL\alpha \leq 7$ dB	0.9	250
A3	$8 \leq DL\alpha \leq 11$ dB	1.0	80
A4	$DL\alpha > 11$ dB	1.0	40

3.3 Vertical extensions

Vertical extensions code information for all objects that are not part of the boundary but that are connected to the control points; i.e. object that can be represented by positions along a vertical line extending above the position encoded in the propagation path.

This includes information about the source, the receiver, thin barriers in the propagation plane, reflections from vertical obstacles and diffractions around vertical edges.

All of the extensions have a <h> tag encoding the height of the extended object above the boundary.

The simplest examples of extensions are the source and the receiver, whose heights are measured relative to the underlying solid.

Note:

> The <z> tag encodes absolute height, i.e. with respect to some external geodesic reference; the <h> tag encodes relative heights, i.e. with respect to the absolute <z> value of the underlying solid (ground or roof).
>
> In case the source or the receiver are positioned on top of a building, <z> specifies the absolute height (or altitude) of the roof (not of the ground under the building) and <h> the height as measured above the roof.

3.4 Coding a simple propagation path

To illustrate the principles described in the previous sections, we construct a simple path lying entirely in the y = 0 plane.

The path includes diffraction over a building. In the propagation plane, the building is represented by three segments, i.e. two vertical segments for the façades on either side and one vertical segment for the roof. Additional segments represent the ground between the source and the building and between the building and the receiver. Material class A2 is used to represent the fact that the building is covered by a green roof…

```
<path>
    <!-- footprint of the source at (x=0, z=0) -->
    <!-- height = 5 cm above local ground -->
    <cp>
      <pos>
         <x> 0.00 </x>
         <y> 0.00 </y>
         <z> 0.00 </z>
       </pos>
       <mat id=”D” />
       <ext>
           <source>
              <h> 0.05 </h>
           </source>
       </ext>
    </cp>
    <!-- ground between source and building -->
    <!-- ground type = D  -->
    <cp>
       <pos>
           <x> 25.00 </x>
           <z>  0.00 </z>
       </pos>
       <mat id=”D” />
    </cp>
    <!-- vertical wall, material type = A0 -->
    cp>
       <pos>
```

```
            <x> 25.00 </x>
            <z>  6.50 </z>
         </pos>
         <mat id="A0" />
      </cp>
      <!-- roof of building, material type = A2 -->
      <cp>
          <pos>
            <x> 35.00 </x>
            <z>  6.50 </z>
          </pos>
          <mat id="A2" />
      </cp>
      <!-- vertical wall, material type = A0 -->
      <cp>
          <pos>
            <x> 35.00 </x>
            <z>  0.00 </z>
          </pos>
          <mat id="A0" />
      </cp>
      <!-- ground behind building, up to receiver -->
      <!-- ground type = D -->
      <!-- receiver height = 4.50m above local ground -->
      <cp>
         <pos>
            <x> 90.00 </x>
            <z>  0.00 </z>
         </pos>
         <mat id="D" />
         <ext>
            <receiver>
               <h> 4.50 </h>
            </receiver>
         </ext>
      </cp>
   </path>
```

Note:

The <source> and <receiver> extensions are mandatory for the first and last control points of the path and not allowed on any intermediate control points. The propagation module accepts both paths from source to receiver or inverse paths from source to receiver, i.e. the path may contain <source> information on the first control point and <receiver> information on the last point or vice-versa.

3.5 Selection and configuration of the calculation method

The minimal requirement for a valid input file is that it contains the mandatory <?xml> header, followed a by a single <CNOSSOS-EU> root entity. The <CNOSSOS-EU> entity in turn must contain a valid propagation path and the specification of a default calculation method.

Although the calculation method can be specified from the command line, the specification of a default method in the input file is mandatory.

A minimal XML file may look like:

```
<?xml version="1.0" encoding="UTF-8" ?>
<CNOSSOS-EU version="1.001">
      <!-- select calculation method -->
      <method>
         <select id="JRC-draft-2010" />
      </method>
      <path>
   ...
      </path>
</CNOSSOS-EU>
```

The test software currently supports three calculation methods: ISO-9613-2, JRC-2012 (also known as NMPB-2008) and JRC-draft-2010 (also known as Harmonoise P2P).

For each method, it is possible to define a specific meteorological model to be used for the evaluation of long-time averaged noise equivalent noise levels. The test software implements two methods for the averaging over meteorological conditions:

- The method described in the ISO 9613-2 standard, based on an heuristic meteorological correction factor C_0 (in dB), which is a direct measure of the difference between the noise level under favourable propagation conditions and the long-time averaged noise level at large distance.

- The method described in the JRC-2012 report, which is based upon the French NMPB-2008 standards and estimates the long-time averaged noise level as a weighted average over homogeneous and favourable propagation conditions. This method requires as input the frequency of occurrence of the favourable propagation conditions.

Although the Harmonoise and Imagine projects proposed a method based on 1 to 5 propagation classes, this model has not been examined in the framework of the CNOSSOS-EU project. For the purpose of strategic noise mapping, it is sufficient to consider two propagation conditions:

- Homogeneous conditions, characterised by an equivalent linear gradient A = 0 s^{-1}.
- Favourable conditions, characterised by an equivalent linear gradient A = 0.07 s^{-1}.

As all three methods can predict noise levels under favourable propagation conditions, they all can be used in conjunction with the ISO-9613-2 meteorological correction term in order to estimate long-term averaged noise levels. The ISO-9613-2 propagation model however cannot be used with the NMPB approach as the ISO method cannot predict noise levels under neutral propagation conditions. It is therefore recommended to specify both the C_0 and the frequency of occurrence of favourable conditions in the input file and to let the calculation method select the most appropriate meteorological characterisation, e.g.:

```
<?xml version="1.0" encoding="UTF-8" ?>
<CNOSSOS-EU version="1.001">
    <!-- select calculation method -->
    <method>
      <select id="JRC-draft-2010" />
      <meteo model="DEFAULT">
          <temperature> 20.0 </temperature>
          <humidity> 60.0 </humidity>
          <pFav> 0.50 </pFav>
          <C0>   3.00 </C0>
      </meteo>
    </method>
    <path>
     ...
    </path>
</CNOSSOS-EU>
```

The meteorological model can take four values:

Value	Meaning
ISO-9613-2	Use the C_0 correction term as defined in ISO-9613-2
NMPB-2008	Use the frequency of occurrence of favourable propagation conditions as defined in the French standard NF 331-133
JRC-2012	Same as NMPB-2008
DEFAULT	Select the meteorological model according to the selected calculation method.

Note that, in the software, the calculation method and the meteorological method can be set independently; i.e. any meteorological model can be used in combination with any of the propagation methods.

- If the ISO-9613-2 meteorological model is uses, the C0 correction will be applied to the noise level under favourable conditions (and in case the JRC-2012 or JRC-draft-2010 propagation models, the noise level under homogeneous propagation conditions will be ignored).
- If, on the other hand, the NMPB-2008 meteorological level is used in combination with the ISO-9613-2 propagation model, the noise level under homogeneous conditions will be set to minus infinity, i.e. it is assumed that homogeneous and unfavourable propagation conditions lead to noise levels that are very low compared to favourable conditions.

3.6 Sound power

If not specified, the calculations are carried for a source that has a sound power output of 0 dB in all octave bands, i.e. the reported output values can be assimilated with a transfer function

between the source and the receiver. In order to calculate sound pressure levels, the sound power of the source has to be defined.

This can be done in two ways: either by making reference to an external file as those produced by the different source models or by entering sound power directly into the input file.

In case of an external file, one only specifies the name of the file that contains the full specification of the source, i.e. the height of the elementary source and its associated sound power are considered part of the source model and therefore provided on output of the respective source modules for road, railway and industrial noise.

```
<ext>
    <!-- use external file -->
    <source>
       <import file="./CNOSSOS_Road_Output.xml" />
    </source>
</ext>
```

Note that the name of the imported source file is relative to the path of the current input file, i.e. in this case the file will be searched for in the folder containing the original input file even in case the program is started from any other folder.

Alternatively, the sound power can be defined directly in the input file:

```
<ext>
   <!-- user defined equivalent source -->
   <!-- an equivalent source is defined in terms of source -->
   <!-- height and sound power output -->
   <source>
      <h> 0.50 </h>
      <Lw sourceType="PointSource"
          measurementType="HemiSpherical"
          frequencyWeighting="LIN">
          80.0  90.0  95.0  100.0  100.0  100.0  95.0  90.0
      </Lw>
   </source>
</ext>
```

The sound power is given as an array of 8 values representing the octave band spectrum ranging from 63 Hz up to 8 kHz. The attributes of the <Lw> tag specify the particular type of sound power encoded:

- The sound power relates to a unitary (point) source (i.e. sound power is relative to the 10^{-12}W standard reference), to a line source (the sound power per unit of length is expressed relative to 10^{-12} W/m) or an area source (the sound power per unit of area is expressed relative to 10^{-12} W/m²).
- According to different standards, sound power can be measured either under free field radiation conditions or for a source placed over a hard reflecting surface. In general, the presence of a hard surface near the source increases its sound power output. Correct assessment of the measurement conditions is important when it comes to coupling

different source and propagation models. This information is a direct consequence of the measurement standard used to assess the sound power of the source. If no information is available, this attribute may be set to "Undefined".

- Sound power spectra (especially when expressed in octave bands) can be reported with or without dB(A) weighting applied. For higher accuracy, dB(A) weighting should be applied before transforming acoustical data into octave bands.

It is possible to combine both methods, e.g. to change the default source height set by the external source models. Note that if this method is used, the <import> tag must appear before the <h> tag.

```
<ext>
   <!-- use sound power as output by the road model -->
   <!-- but set height of the equivalent point source to 30 cm →
   <source>
      <import file="./ CNOSSOS_Road_Output.xml" />
       <h> 0.30 </h>
   </source>
</ext>
```

In this example, the sound power calculated by means of the Cnossos-EU road module will be assigned to an elementary source at 30cm above the road surface (instead of using the 5cm default value specified in the reference documents).

3.7 Thin barriers

A <barrier> tag can be used to place a thin barrier in the propagation path, as in the following example:

```
<path>
    ...
    <!-- footprint of the source at (x=0, z=0) -->
    <!-- height = 5 cm above local ground -->
    <cp>
      <pos>
         <x> 0.00 </x>
         <z> 0.00 </z>
      </pos>
      <mat id="D" /t>
      <ext>
         <source>
            <h> 0.05 </h>
          </source>
      </ext>
    </cp>
    <!-- ground between source and barrier -->
    <!-- ground type = D  -->
    <!-- barrier height = 2m -->
    <!-- barrier material type = A4  -->
    <cp>
      <pos>
          <x> 25.00 </x>
```

```
            <z>  2.00 </z>
          </pos>
          <mat id="D" />
          <ext>
            <barrier>
              <h> 2.00 </h>
              <mat id="A4" />
            </barrier>
          </ext>
        </cp>
        <!-- ground behind the barrier, up to receiver -->
        <!-- ground type = D -->
        <!-- receiver height = 4.50m above local ground -->
        <cp>
          <pos>
            <x> 90.00 </x>
            <z>  5.00 </z>
          </pos>
          <mat id="D" />
          <ext>
            <receiver>
              <h> 4.50 </h>
            </receiver>
          </ext>
        </cp>
      </path>
```

The material covering the faces of the barrier is encoding using the <mat> property, similar to the segments that make up the boundary in the propagation path. Note that the material property of the barrier's faces will be ignored in the ISO-9613-2 or JRC-2012 (NMPB) methods but that the JRC-Draft-2010 (Harmonoise) method takes this into account and produces a slightly lower level in case the barrier is covered with an absorbing material.

Note that the <barrier> tag is a convenience function and that we might have encoded this path using two additional positions:

```
      <path>
        ...
        <!-- ground between source and barrier -->
        <cp>
          <pos>
            <x> 25.00 </x>
            <z>  2.00 </z>
          </pos>
          <mat id="D" />
        </cp>
        <!-- source side of the barrier -->
        <cp>
          <pos>
            <x> 25.00 </x>
            <z>  4.00 </z>
          </pos>
          <mat id="A4" />
        </cp>
```

```
        <!-- receiver side of the barrier -->
        <cp>
          <pos>
              <x> 25.00 </x>
              <z>  2.00 </z>
          </pos>
          <mat id="A0" />
        </cp>
        ...
    </path>
```

Note that the multiple-segment representation of a barrier offers extra flexibility, e.g. in case the two faces of the barrier are covered with different materials, or if the ground has a different altitude on either side of the barrier. Thick barriers (such as buildings) can only be encoded using the multiple-segment representation.

3.8 Reflected paths

Reflections from vertical obstacles are encoded using the <wall> extension tag. The tag has two attributes for encoding the height of the reflecting obstacle above local ground and the acoustical properties of the material covering the reflecting wall.

Example for a reflection from an obstacle aligned with the y-axis at x = 20m:

```
    <path>
        ...
        <!-- footprint of the source at (x=0, y=0, z=0) -->
        <!-- height = 5 cm above local ground -->
        <cp>
           <pos>
              <x> 0.00 </x>
              <y> 0.00 </y>
              <z> 0.00 </z>
           </pos>
           <mat id="D" />
           <ext>
              <source>
                <h> 0.05 </h>
              </source>
           </ext>
        </cp>
        <!-- ground between source and reflecting obstacle  -->
        <!-- ground type = D  -->
        <!-- specular reflection from a vertical wall in the plane x = 20 -->
        <!-- wall covered with material type = A3 -->
        <cp>
          <pos>
             <x> 20.00 </x>
             <y> 10.00 </y>
             <z>  0.00 </z>
          </pos>
          <mat id="D" />
          <ext>
              <wall>
```

```
                <h> 5.00 </h>
                <mat id="A3" />
              </wall>
          </ext>
        </cp>
        <!-- ground between reflection obstacle and receiver -->
        <!-- ground type = D -->
        <!-- receiver height = 4.50m above local ground -->
        <cp>
          <pos>
              <x> -20.00 </x>
              <y>  30.00 </y>
              <z>   0.00 </z>
          </pos>
          <mat id="D" />
          <ext>
              <receiver>
                <h> 4.50 </h>
              </receiver>
          </ext>
      </cp>
    </path>
```

Notes:

The material property of the reflecting wall is used to determine the attenuation due to absorption. The relative property is the absorption coefficient α_S associated with the material identifier. For all predefined materials, this property is derived from the acoustical impedance. This property is exploited identically in all three propagation methods. Alternatively, the absorption coefficients of the material can be defined by the end-user (see next section).

The height of the reflecting obstacle is used to check whether the specular reflection points falls within the finite (vertical) size of the reflector (e.g. in the ISO 9613-2 method) and/or to calculate the retro-diffraction term (e.g. in the JRC-2012 method).

Limitations:

In order to confirm strictly to the ISO-9613-2 standard we would need to know the 3D extent of the reflecting barrier (see e.g. Figure 8 of the standard) in order to implement eq. 19. As a first approximation, we consider that the length (i.e. the horizontal extent) of obstacles is typically much larger than their height and that we may assume l_{min} = h.

A more complete treatment might be implemented in a future version of the software, but this will require the complete 3D geometry of the reflector being passed as input to the calculation modules, e.g. as part of the <reflection> item. Knowing that most path finders may have problems finding the horizontal extent of a reflecting obstacle (e.g. because it is segmented in small parts), it is questionable whether this is worth the effort (especially for the purpose of strategic noise mapping).

The projection of the path and the reflecting wall on a horizontal plane should obey Snell's laws of specular reflection (e.g.in the example below the reflecting wall is located

in the plane x = 20). This is not checked by the Cnossos-EU library, as it is assumed that, in operational software, conformity of the path can be guaranteed by the path finder (i.e. because the algorithm generates specular reflected ray paths in 2D).

3.9 Lateral diffractions

Propagation paths laterally diffracted around vertical edges are coded similarly to reflected paths, using the <edge> extension tag.

Example of a path diffracted by a vertical edge at x = 20, y = 15.

```
<path>
    <!-- footprint of the source at (x=0, y=0, z=0) -->
    <!-- height = 5 cm above local ground -->
   <cp>
      <pos>
          <x> 0.00 </x>
          <y> 0.00 </y>
          <z> 0.00 </z>
      </pos>
      <mat> D </mat>
      <ext>
          <source>
             <h> 0.05 </h>
          </source>
      </ext>
    </cp>
    <!-- ground between source and diffracting edge  -->
    <!-- ground type = D  -->
    <!-- lateral diffraction by a vertical edge-->
    <cp>
      <pos>
          <x> 20.00 </x>
          <y> 15.00 </y>
          <z>  0.00 </z>
      </pos>
      <mat> D </mat>
      <ext>
          <edge>
             <h> 5.00 </h>
          </edge>
      </ext>
    </cp>
    <!-- ground between edge and receiver -->
    <!-- ground type = D -->
    <!-- receiver height = 4.50m above local ground -->
    <cp>
      <pos>
```

```
            <x>  40.00 </x>
            <y>  25.00 </y>
            <z>   0.00 </z>
        </pos>
        <mat> D </mat>
        <ext>
            <receiver>
               <h> 4.50 </h>
            </receiver>
        </ext>
    </cp>
</path>
```

Notes

A laterally diffracting edge has a finite height. Although none of the methods mentions how this information is to be used, we may assume that lateral diffraction is limited to the case where the diffraction point falls within the finite extent of the vertical edge (where it is assumed that the diffraction point is defined by Keller's law of diffraction).

Advanced modelling of lateral diffraction might need extra information on the faces of the wedge joining at the vertical edge (e.g. the wedge angle and the materials covering the faces of the wedge are important properties that influence the diffraction effects). Such effects are ignored in all three methods considered in this project, therefore no material properties are assigned to the diffracting edge.

3.10 Material properties

Materials enter the acoustical calculations through a set of associated properties. In order to guarantee coherence of these properties, the software makes a distinction between "materials" and "properties" associated with each material. For each material, the following properties are stored and/or calculated:

- The ground factor G as defined in ISO 9613-2
- The flow resistivity parameter
- The specific acoustical impedance spectrum
- The diffuse field absorption coefficient spectrum

The properties of materials are stored and maintained in a shared database inside the software. End-users can add new materials to this database and change the acoustical properties associated with materials.

The minimal requirement for each material is to have a ground factor G defined. All other properties are optional and can be either defined in terms of user-defined values or, if missing, evaluated in terms of the other properties, see Appendix E.4.

The Cnossos-EU software allows for new materials to be defined in the XML input files. Material definitions must occur before the propagation path that uses them.

The <materials> tag can contain any number of <mat> child tags. If the <mat> identifier specifies an unknown name, a new material record will be created; otherwise the <mat> record will modify the properties associated with one of the predefined materials.

Within the <mat> tag, new values of the acoustical properties are set by means of the <G>, <sigma> and <alpha> tags. All properties are optional. Missing properties will keep their predefined (eventually undefined) values.

The previous example can be adapted to use user-defined absorption coefficients for the reflecting wall and to use a slightly modified ground factor for the predefined material class G:

```
<CNOSSOS-EU>
    ...
    <materials>
      <!-- user defined material properties -->
       <mat id = "UserDefined">
           <G> 0.9 </G>
           <alpha>
              0.12  0.22   0.36   0.45  0.58   0.60   0.55   0.52
           </alpha>
       </mat>
      <!-- change properties of predefined material-->
       <mat id = "G">
           <G> 0.1 </G>
       </mat>
    </materials>
    <!-- propagation path -->
    <path>
    ...
    <!–reflection from wall covered with user defined material -->
        <cp>
          <pos>
             <x> 20.00 </x>
             <y> 10.00 </y>
             <z>  0.00 </z>
          </pos>
          <ext>
              <wall>
                 <h> 5.00 </h>
                 <mat id=”UserDefined” />
              </wall>
          </ext>
        </cp>
        ...
    </path>
</CNOSSOS-EU>
```

3.11 Extended source geometry

Excess attenuation is always calculated assuming point-to-point propagation. However, calculation of the free-field response of the source may include other effects:

- Orientation of the source is a necessary input in order to evaluate the directional sound power of the source, i.e. the sound power radiated in a given direction taken into account the source's directivity.
- For a source line of finite length, more accurate results can be obtained by analytically integrating the spherical spread over the source line segment. Modern path finder algorithms do not replace such segments by equivalent point sources but rely on inverse ray or beam tracing in order to construct homogeneous propagation sectors between a point receiver and a line source segment of finite length. Such algorithms are generally faster than brute force methods that split the source into equally spaced point sources.

The CNOSSOS-EU propagation library includes support for both features by means of an extended source description. The <extGeometry> is used to encode the specific geometrical and dimensional attributes for different types of sources, it has support for point, line and area sources in general and for segmented source lines particularly.

3.11.1 Point source

A point source contains additional information on the orientation of the source with respect to the project's coordinate system. This information is used to construct a local coordinate system relative to the oriented source and then to convert the absolute direction of propagation into a direction relative to the local (project independent) coordinate system of the elementary source. Elementary sources are assumed to be able to evaluate directivity and apparent sound power in local (project independent) coordinates. Local coordinate systems and conversion of directional vectors are explained in appendix E.

The example below defines a point source oriented in the direction of the X-axis and 45° tilted downwards. Note that the directional vector is not normalised on input but that the software takes care of this…

```
<path>
    <cp id="source">
        <pos>
            <x>   0.00 </x>
            <y>   0.00 </y>
            <z>   0.00 </z>
        </pos>
        <mat id="H" />
        <ext>
            <source>
               <h> 0.50 </h>
               <extGeometry>
                  <pointSource>
                     <orientation>
                        <x>  1.0 </x>
                        <y>  0.0 </y>
                        <z> -1.0 </z>
                     </orientation>
                  </pointSource>
               </extGeometry>
            </source>
```

```
            </ext>
        </cp
        ...
    </path
```

3.11.2 Line source

A line source is defined similarly to a point source. It is up to the application to replace the line source by an equivalent point source and to pass in the length of the original line source. In principle, this is done while pre-processing the geometrical input and splitting extended source line into small segments. Each segment is then replaced by an equivalent point source and the length of the segment used to evaluate the sound power associated with the elementary source. The Cnossos-EU library can take care of this in a unified way. Apart the orientation, the line source element has an additional attribute for storing the length of the original source segment.

```
    <path>
        <cp id="source">
            <pos>
                <x>   0.00 </x>
                <y>   0.00 </y>
                <z>   0.00 </z>
            </pos>
            <mat id="H" />
            <ext>
                <source>
                   <import file="CNOSSOS_Road_Output.xml" />
                   <extGeometry>
                       <lineSource>
                          <length> 10.0 </length>
                           <orientation>
                               <x>  0.8 </x>
                               <y> -0.6 </y>
                               <z>  0.0 </z>
                            </orientation>
                       </lineSource>
                    </extGeometry>
                </source>
            </ext>
        </cp
        ...
    </path
```

Note that in general, the orientation of a line source is taken horizontal and perpendicular to the source line. E.g. if the source line extends from (x=0, y=0) towards (x=300,y=400), the normalised orientation is given by the vector (x = 0.8, y=-0.6).

Note that, unless the "CheckSoundPowerUnits" is set to false, the library will check that the sound power defined in the external source file will be compatible with the source geometry, i.e. that sound power is defined per unit of length.

In the output table, the length (L) of the source segment is integrated as part of the geometrical spread , so that the first line always contains the sound power as found in the input files. I.e. the geometrical spread is defined as:

$$\Delta L_{geo} = 10.log\frac{L}{4\pi d^2}$$

3.11.3 Area source

An area source is encoded similarly to a line source but its dimensions are specified by means of an <area> attribute instead of length. As above, the software will check that the sound power associated with this source actually has the units of sound power per unit of area.

In general, the orientation of area sources is taken equal to the normal to the surface supporting the area source (e.g. the normal vector of the vertical wall for a façade mounted source or the Z-axis for a roof-mounted source).

```
<path>
    <cp id="source">
        <pos>
            <x>   0.00 </x>
            <y>   0.00 </y>
            <z>   0.00 </z>
        </pos>
        <mat id="H" />
        <ext>
            <source>
               <h> 0.05 </h>
               <Lw sourceType=”AreaSource” measurementType=”HemiSpherical”>
                    98.0 95.6 89.6 88.6 90.2 88.4 87.4 82.9
               </Lw>
               <extGeometry>
                   <areaSource>
                     <length> 10.0 </length>
                      <orientation>
                          <x>  0.8 </x>
                          <y> -0.6 </y>
                          <z>  0.0 </z>
                       </orientation>
                   </areaSource>
                </extGeometry>
            </source>
        </ext>
    </cp
    ...
</path
```

Similarly, the geometrical spread of an area source is calculated as:

$$\Delta L_{geo} = 10.log\frac{A}{4\pi d^2}$$

3.11.4 Line source segments

A line source segment is explicitly defined by two points. In case of a line source segment, the geometrical spread is calculated as:

$$\Delta L_{geo} = 10.log\frac{\Delta\alpha}{4\pi R}$$

Where $\Delta\alpha$ is the angle of view of the segment as seen from the (image) receiver and R the shortest distance from the (image) receiver to the infinite line containing the segment.

Typically, advanced path finding algorithms (such as beam tracers or image methods) can be used to explicitly construct homogeneous propagation sectors between a receiver positions and a source line. Such a sector is delimited by two vertical planes and contains a segment of the line source defined explicitly by its endpoints. For the calculation of the excess attenuation, a single propagation plane will be constructed, generally passing through the segment's midpoint. The Cnossos-EU propagation library can use this information on input and evaluate the geometrical accordingly. It should be noted that this method is more accurate than the simple fixed-length line source model described above.

```
<path>
    <!-- source -->
    <cp id="source">
      <pos>
          <x>  0.00 </x>
          <y>  0.00 </y>
          <z>   0.00 </z>
      </pos>
      <mat id="H" />
      <ext>
          <source>
            <import file="CNOSSOS_Road_Output.xml" />
            <!-- source geometry in the horizontal plane
                 = segment from Y=-10 to Y=+10m -->
            <extGeometry>
                <lineSegment>
                  <posStart>
                     <x>   0 </x>
                     <y> -10 </y>
                     <z>   0 </z>
                  </posStart>
                  <posEnd>
                     <x>   0 </x>
                     <y>  10 </y>
                     <z>   0 </z>
                  </posEnd>
                </lineSegment>
            </extGeometry>
          </source>
      </ext>
    </cp>
    ...
</path>
```

Optionally, the same integration method can be used with fixed angle (inverse) ray-tracing. In this method, the path finder construct a fixed number N of discrete ray paths from the receiver; each ray being considered the bisector of an implicit angular sector with an opening angle of 360/N degrees.

Such a ray path intersects a segment of a source line in a single point, encoded as the propagation path's equivalent source position. It is implicitly assumed that the intersection extends symmetrically on both sides of the central path (by half the opening angle associated with the central ray path) and forms a homogeneous propagation sector.

As above, the library can use this information on input and evaluate the geometrical spread integrated over the angular sector more accurately than in the case of fixed-length source line segmentation.

```
<path>
    <!-- source -->
    <cp id="source">
      <pos>
          <x>  0.00 </x>
          <y>  0.00 </y>
          <z>   0.00 </z>
      </pos>
      <mat id="H" />
      <ext>
          <source>
            <import file="CNOSSOS_Road_Output.xml" />
            <!-- source geometry in the horizontal plane
                 = straight line from Y=-1000 to Y=+1000m -->
            <extGeometry>
                <lineSegment>
                  <posStart>
                     <x>     0 </x>
                     <y> -1000 </y>
                     <z>   -25 </z>
                  </posStart>
                  <posEnd>
                     <x>     0 </x>
                     <y>  1000 </y>
                     <z>    25 </z>
                  </posEnd>
                   <fixedAngle> 2.0 </fixedAngle>
                </lineSegment>
            </extGeometry>
          </source>
      </ext>
    </cp>
    ...
</path>
```

Note that in the last example, the segment's end-points refer to the original data for the source line as a whole, not to the intersection of this line with the homogeneous propagation sector.

Note that, in discrete ray-tracing techniques, the actual intersection of the angular sector with the source line is never explicitly evaluated and this may lead to some discretisation errors as the propagation plane intersects the line near one of its endpoints.

3.12 Valid propagation paths and options

A direct path is shown in Figure VI.1 of JRC-2012. Such a path is lying in a single vertical plane. Therefore, when projected on a horizontal plane, a direct path must form a single straight line segment. Moreover, in order to represent sound propagation in a given direction, the sequence of control points must be ordered in increasing (horizontal) distance from source to receiver (or vice-versa).

Such pre-conditions (and others) are systematically checked by the software and in case this validation check (or any other) fails, the test software will return an appropriate "invalid input data" message and the attenuations will be undefined.

This kind of conformity checks may be redundant in case the propagation paths are constructed by means of an automated path finder algorithm; i.e. most algorithms can guarantee that the conformity conditions are met as a side-effect of the way the path is constructed.

In order to suppress the conformity checking, one can use options:

```
<?xml version="1.0" encoding="UTF-8" ?>
<CNOSSOS-EU version="1.001">
       <!-- select calculation method -->
       <method>
          <select id="JRC-draft-2010" />
           <options>
               <option id="CheckHorizontalAlignment" value="false" />
           </option>
       </method>
       <path>
   ...
       </path>
</CNOSSOS-EU>
```

The complete list of available options is given in the table below. Some options are mandatory when using a particular propagation method; setting or clearing such options in the input file will be ignored. Options can also be used to reinforce the comparability of methods; i.e. by disabling features that are supported in some methods but not in all.

Additional information on options and how they modify the behaviour of the software can be found in the next section, i.e. in the test cases that were used to verify the expected behaviour of the software when different options are set or unset.

Option	Meaning
ForceSourceToReceiver	The Cnossos-EU library accepts propagation paths running from source to receiver or vice-versa. If this option is set, paths running from receiver towards the source will be reversed before calling any acoustical calculation.
CheckHorizontalAlignment	When projected on the horizontal plane, a propagation path must correspond to a set of straight line segments, changes of direction may occur only at positions corresponding to reflections from vertical walls or diffractions around vertical edges. Moreover, control points must appear in increasing

	distance of propagation. Setting this option will disable the checking of this requirement, e.g. in case the use of an automated path finder algorithm ensures that these requirements will always be met.
CheckLateralDiffraction	Attenuations due to lateral diffraction are generally calculated using formulas derived for diffraction over obstacles. This implies that the laterally diffracted path must have a geometry that, when projected on the horizontal plane, represent a problem similar to the canonical problem of diffraction by or more obstacles in the vertical plane. In order to enforce this requirement, the software checks that the horizontal projections of diffracted path from source to receiver, closed by the direct path from receiver to source, form a convex polygon.
DisableReflections	Ignore paths that contain one or more reflections from vertical obstacles.
DisableLateralDiffractions	Ignore paths that contain one or more diffractions by vertical edges.
CheckHeightLowerBound	If this option is set, the software checks that the specular reflection points are located above the foot points of the reflecting obstacles. If the option is not set, back-scattered ray paths (i.e. a combination of reflections and retro-diffraction) are allowed. A similar approach is taken for the top of the Keller cone in case of lateral diffraction by a vertical edge.
CheckHeightUpperBound	If this option is set, the software checks that the specular reflection points are located below the top position of the reflecting obstacles. If the option is not set, back-scattered ray paths (i.e. a combination of reflections and retro-diffraction) are allowed. A similar approach is taken for the top of the Keller cone in case of lateral diffraction by a vertical edge.
CheckSoundPowerUnits	If this option is set, the software checks that the sound power is defined in terms of sound power units that are compatible with the extended source geometry. E.g. for a line source, the source strength is defined as sound power per meter of source and must be expressed relative to 10^{-12}W/m.
SimplifyPathGeometry	Explicit simplification of the path geometry was first introduced in the Nord-2000 projects and considered part of the Harmonoise model. Setting this option may help reducing computation time. Note that simplification of the input data has limited effect when using the ISO-9613-2 or JRC-2012 methods because these methods (explicitly or implicitly) take into account some approximation of the ground plane anyhow.
IgnoreComplexPaths	Complex paths may contain lateral diffraction in combination with reflections from vertical obstacles or diffractions over horizontal edges. If this option is set, complex paths will be ignored. Note that none of the implemented methods can handle complex paths but this may change in future versions.
ExcludeGeometricalSpread	Do not include attenuation due to geometrical spread. This is useful in case the application software already contains code for determining the geometrical spread, e.g. when this is implemented in common software on top of different propagation methods.
ExcludeAirAbsorption	Do not include atmospheric absorption in the acoustical calculations. This is useful in case the application software already contains code for determining

	the atmospheric absorption, e.g. when this is implemented in common software on top of different propagation methods.
ExcludeSoundPower	Do not include sound power in the acoustical calculations, i.e. the result will represent the transfer function between the source power and the level at the receiver. This is useful if the application software has his own code for determining the sound power of different sources and/or if it needs to store sound power and attenuation separately.

The table below lists the default settings of the options and the mandatory settings imposed by the different propagation models. As the test software is mainly intended to be used for manually constructed paths, all conformity checks are enabled unless otherwise stated in the input file. Note that, for normal use of the methods (i.e. respecting strict compliance with the reference texts), it is not necessary (nor recommended) to specify any options in the input file.

Option	Default value	ISO-9613-2	JRC-2012	JRC-draft-2010
ForceSourceToReceiver	false		true	
CheckHorizontalAlignment	true			
CheckLateralDiffraction	true	true	true	n.a.
DisableReflections	false			
DisableLateralDiffractions	false			true
CheckHeightLowerBound	false	true		
CheckHeightUpperBound	false	true		
CheckSoundPowerUnits	true			
SimplifyPathGeometry	false			
IgnoreComplexPaths	true	true	true	n.a.
ExcludeGeometricalSpread	false			
ExcludeAirAbsorption	false			
ExcludeSoundPower	false			

4 Validation

The CNOSSOS-EU software has been intensively tested in order to detect and correct as many errors as possible. It must be understood that these validations solely apply to the implementation of the different propagation methods in software and do not, in any way, evaluate the accuracy of the predicted results. However, the tests may throw light on the consequences and limitations due to the various modelling approaches adopted in different methods.

4.1 Plausible results

A first series of tests was specifically designed to check the plausibility (not the accuracy) of the results produced by the different methods. Non-plausible results may be due to programming errors or to incomplete or incoherent specifications in the reference texts. Programming errors have been corrected by the author of the software. Incomplete and/or incoherent specifications have been reported in the issue logs and appropriate solutions have been proposed by the Consortium and approved by the Noise Experts designated by the Member States.

To judge the plausibility of the calculated results, the following "well known facts" have been taken into consideration:

- Noise levels decrease by approx. 6 dB when doubling the propagation distance,
- Noise levels are higher under favourable propagation conditions,
- Meteorological effects increase with distance,
- Noise levels decrease when the height of a barrier is increased,
- Barriers not blocking the line of sight from the source to the receiver have limited efficiency,
- All methods produce comparable results; i.e. differences in calculated results are of the same order of magnitude as the uncertainty of measured data under similar conditions.

The names of the following sections refer to the file names as listed in section 2 of this document.

All tests consider a single source and up to 6 receivers at increasing distance from the source. The source has a fixed sound power spectrum as shown in the table below.

Freq (Hz)	63	125	250	500	1000	2000	4000	8000
Lw (dB)	80	90	95	100	100	100	95	90

For all tests, meteorological effects on the long-time averaged equivalent noise level are accounted for by considering 50% of favourable propagation conditions, or equivalently C_0 = 3 dB.

For all cases, a mix of soft and hard ground has been used; hard ground being assimilated to class H, soft ground to class D.

Each test case has been calculated by means of all three calculation methods without any modification of the input file.

4.1.1 Flat ground

The propagation contains an impedance jump between a 5m hard strip (G=0) on the source side and a soft ground (G=1) on the receiver side. The source height is set to 50cm, the receiver is located at 2.5m above the terrain.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	75.1	68.2	59.3	52.3	45.3	35.0
	Lp,H	-	-	-	-	-	-
	Leq	75.1	68.2	58.1	50.2	42.7	32.3
JRC-2012	Lp,F	76.8	70.4	60.3	51.8	45.2	36.9
	Lp,H	76.8	70.4	61.3	49.3	37.7	19.4
	Leq	76.8	70.4	60.8	50.8	42.9	34.0
JRC-draft-2010	Lp,F	76.5	69.8	60.2	52.6	46.5	40.6
	Lp,H	76.5	69.8	60.0	50.7	40.4	30.3
	Leq	76.5	69.8	60.1	51.7	44.5	37.9

Note: for all tests, meteorological effects on the long-time averaged equivalent noise level are accounted for by considering 50% of favourable propagation conditions, or equivalently C_0 = 3 dB.

As expected, the ISO 9613-2 method does not allow calculations under homogeneous propagation conditions. As expected, all methods produce comparable results with differences of approx. 1 dB(A) at short distances, increasing with distance, up to 5 dB(A) at 500m.

4.1.2 Embankment, h = 1m

A 5 cm high source is placed on a hard platform which is connected to the flat terrain by means of a 2m wide, 1m high sloping ground plane. The receiver is located at 2.5m height relative to the flat terrain.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	76.2	69.8	61.3	54.8	48.0	38.2
	Lp,H	-	-	-	-	-	-
	Leq	76.2	69.8	59.8	52.6	45.4	35.3
JRC-2012	Lp,F	76.8	70.5	60.8	52.7	46.0	36.9
	Lp,H	76.8	70.5	61.4	51.5	41.9	24.3
	Leq	76.8	70.5	61.1	52.1	44.4	34.1
JRC-draft-2010	Lp,F	78.7	72.3	62.0	55.9	50.0	42.4
	Lp,H	78.8	72.3	61.9	54.3	44.4	33.3
	Leq	78.7	72.3	62.0	55.2	48.0	39.9

Differences are slightly larger than in the flat case, even at shorter distances. This test case illustrates the fact that different propagation methods use different strategies to predict the effect of a wedge-shaped diffracting terrain edge near the source.

4.1.3 Embankment, h = 2m

Similar the previous test case but using a 2m high, 3m wide slope.

Method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	76.3	69.8	61.3	54.8	48.0	38.2
	Lp,H	-	-	-	-	-	-
	Leq	76.3	69.8	59.8	52.6	45.4	35.3
JRC-2012	Lp,F	76.1	67.5	61.3	53.6	46.1	36.9
	Lp,H	76.1	67.4	61.4	53.4	44.6	29.9
	Leq	76.1	67.4	61.4	53.5	45.4	34.7
JRC-draft-2010	Lp,F	79.1	69.5	60.8	55.0	50.0	42.4
	Lp,H	79.0	69.4	60.5	54.7	47.1	35.5
	Leq	79.1	69.5	60.6	54.9	48.8	40.2

As expected, the differences at short distances increase and now reach almost 3 dB(A) at the nearest receiver. Acoustical levels near the shadow boundary of a diffracting edge are rapidly varying with receiver height and therefore hard do measure and to predict.

4.1.4 Embankment, h = 4m

Similar the previous test case but using a 4m high, 5m wide slope. The receiver height has been increased to 5m in order to preserve line-of-sight propagation.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	76.3	69.8	61.4	54.9	48.2	38.4
	Lp,H	-	-	-	-	-	-
	Leq	76.3	69.8	61.4	53.4	45.9	35.7
JRC-2012	Lp,F	76.9	69.6	60.0	55.2	47.4	36.9
	Lp,H	76.9	69.6	59.5	55.0	47.1	35.9
	Leq	76.9	69.6	59.8	55.1	47.2	36.5
JRC-draft-2010	Lp,F	79.0	71.0	60.9	54.6	48.6	41.5
	Lp,H	79.0	71.0	60.7	54.0	47.8	37.7
	Leq	79.0	71.0	60.8	54.3	48.2	40.0

Results and conclusions are similar to the previous case. The only significant change in results occurs at the 20m distance receiver where the JRC-draft-2010 methods seems to emphasis the focusing effect of the lower edge of the sloped terrain.

4.1.5 Depressed, h = 2m

Once again the source is located at 5cm above a 5m wide hard platform, slightly below the level of the flat terrain and connected to it by means of a 2m wide, 1m high upward sloping ground plane. The receiver is located at 2.5m height above the local terrain.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	70.8	61.5	49.0	41.9	35.5	27.5
	Lp,H	-	-	-	-	-	-
	Leq	70.8	61.5	47.5	39.6	32.9	24.7
JRC-2012	Lp,F	76.1	61.9	46.9	37.3	31.0	24.0
	Lp,H	76.1	61.8	46.9	35.1	24.0	8.6
	Leq	76.1	61.8	46.9	36.4	28.7	21.1
JRC-draft-2010	Lp,F	76.5	61.5	44.9	34.5	27.1	24.3
	Lp,H	76.5	61.4	44.5	33.8	25.9	18.4
	Leq	76.5	61.5	44.7	34.2	26.6	22.3

At larger distances all methods lead to comparable results. However, for the receiver nearest to the source, the ISO 9613-2 method predicts a noise level that is significantly different from the others. Such a large difference illustrates the fact that the ISO-9613-2 method is based on a different (more simplified) approach for dealing with diffracting edges not blocking the line of sight between the source and the receiver. This problem is worth being investigated more thoroughly and the ISO-9613-2 method might need some clarification and/or adaptation in order to deal more accurately with such transitions.

4.1.6 Depressed + barrier, h = 1 + 2m

The terrain is similar to the previous case but a 2m high thin barrier is erected on top of the slope.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	62.3	51.8	42.5	36.0	29.6	21.4
	Lp,H	-	-	-	-	-	-
	Leq	62.3	51.8	41.0	33.7	27.0	18.6
JRC-2012	Lp,F	62.4	51.4	41.8	34.3	27.2	19.0
	Lp,H	62.4	51.4	41.8	33.9	25.4	10.4
	Leq	62.4	51.4	41.8	34.1	26.4	16.6
JRC-draft-2010	Lp,F	61.3	50.8	42.1	36.3	31.3	24.9
	Lp,H	61.5	50.7	42.0	36.0	28.7	20.1
	Leq	61.4	50.8	42.1	36.1	30.1	23.1

All receivers are located in the shadow zone behind the barrier. Although the methods use different formulations to deal with diffraction over obstacle, they produce comparable results with differences increasing with distance.

4.1.7 Barrier, h = 2m

In order to evaluate the diffraction effect separately, a 2m high thin barrier is placed at 10m from the source on a flat terrain. The source is located at 50 cm above a hard platform on one side of the barrier, the receiver is located at 2.5m above soft ground on the other side of the

barrier. There is no receiver at 10m distance because its position would coincide with the barrier.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	-	61.0	50.9	44.5	38.4	30.6
	Lp,H	-	-	-	-	-	-
	Leq	-	61.0	49.7	42.4	35.9	27.8
JRC-2012	Lp,F	-	61.6	50.6	43.3	36.3	27.7
	Lp,H	-	61.6	50.5	42.6	34.1	19.1
	Leq	-	61.6	50.5	42.9	35.3	25.3
JRC-draft-2010	Lp,F	-	58.1	47.6	41.2	35.8	29.6
	Lp,H	-	58.1	47.4	40.8	33.1	23.1
	Leq	-	58.1	47.5	41.0	34.7	27.5

At larger distances, all methods produce comparable results. Nearer to the barrier, the JRC-draft-2010 method predicts significantly higher effects for the barrier than the other methods.

4.1.8 Barrier, h = 4m

This case is similar to the previous one with the height of the barrier increased to 4m.

method	level	**10m**	**20m**	**50m**	**100m**	**200m**	**500m**
ISO-9613-2	Lp,F	-	55.4	45.0	38.4	32.2	24.2
	Lp,H	-	-	-	-	-	-
	Leq	-	55.4	45.0	36.9	29.9	21.7
JRC-2012	Lp,F	-	55.2	44.5	37.6	30.8	21.5
	Lp,H	-	55.2	44.4	37.5	30.6	19.3
	Leq	-	55.2	44.4	37.5	30.7	20.5
JRC-draft-2010	Lp,F	-	52.6	42.5	36.7	31.3	24.7
	Lp,H	-	52.6	42.4	36.4	31.3	22.3
	Leq	-	52.6	42.5	36.6	31.3	23.7

As previous, the JRC-draft-2010 method predicts significantly higher effects for the barrier than the other methods for receivers up to 50m behind the barrier. At larger distances, the method predicts slightly higher noise levels probably because it explicitly accounts for turbulent scattering into the shadow zone, whereas this effect is implicit in the other methods (using a fixed limit of -20 or -25 dB maximum attenuation in the deep shadow).

4.1.9 Barrier, h = 1m

This case is similar to the previous ones but the height of the barrier is now reduced to 1m.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	-	65.4	54.0	47.1	40.7	31.9
	Lp,H	-	-	-	-	-	-
	Leq	-	65.4	52.5	44.9	38.1	29.0
JRC-2012	Lp,F	-	68.8	55.8	48.1	41.0	32.1
	Lp,H	-	68.7	55.4	46.1	36.1	18.8

	Leq	-	68.8	55.6	47.2	39.2	29.3
JRC-draft-2010	Lp,F	-	71.7	55.1	48.1	41.9	35.7
	Lp,H	-	71.7	54.8	46.7	36.5	26.2
	Leq	-	71.7	55.0	47.4	40.0	33.2

Although low barriers are often cited as difficult cases with respect to the range of application of simplified prediction schemes, all methods provide more or less comparable results for larger distances. A significant difference occurs at the nearest receiver where the ISO 9613-2 method predicts a significantly lower noise level (see 4.1.5).

4.1.10 Valley, depth = 10m

This case simulates propagation over a complex valley-shaped terrain. The terrain profile in the propagation plane is given by:

distance (m)	0	5	20	50	100	200	500
altitude (m)	10	10	0	10	20	30	40

In this configuration, nearby receivers are in direct view from the receiver whereas receivers further away gradually move into the shadow zone formed by the smooth terrain.

method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	-	69.2	61.3	54.7	43.2	31.9
	Lp,H	-	-	-	-	-	-
	Leq	-	69.2	59.8	52.5	40.6	29.0
JRC-2012	Lp,F	-	70.2	62.5	55.6	44.0	18.8
	Lp,H	-	70.2	62.5	55.6	41.4	14.2
	Leq	-	70.2	62.5	55.6	42.9	17.1
JRC-draft-2010	Lp,F	-	71.9	64.0	57.9	49.1	19.7
	Lp,H	-	72.0	64.1	59.4	45.1	18.5
	Leq	-	71.9	64.0	58.8	47.6	19.1

For all distances except deep in the shadow (i.e. at 500m distance), the methods produce results that are significantly different and increasing with distance. The way a complex terrain is approximated (or not) by an equivalent mean plane clearly influences the calculated results.

4.1.11 High source, h = 4m

This test case considers a 4m high source (typical for industrial sources) on a 10m wide strip of ground and a 1.5m high receiver on soft ground. It is expected that ground and meteorological effects diminish with the height of the source.

Method	level	10m	20m	50m	100m	200m	500m
ISO	Lp,F	76.7	69.2	59.7	52.7	45.8	35.8
	Lp,H	-	-	-	-	-	-
	Leq	76.7	69.2	59.7	51.4	43.6	33.1
JRC-2012	Lp,F	76.8	70.7	62.0	54.4	46.3	37.0
	Lp,H	76.8	70.7	62.0	54.7	45.4	31.4

	Leq	76.8	70.7	62.0	54.6	45.9	35.0
JRC-draft-2010	Lp,F	76.0	68.8	61.1	55.2	49.6	41.7
	Lp,H	76.0	68.8	61.1	55.1	47.6	35.7
	Leq	76.0	68.8	61.1	55.2	48.8	39.6

For shorter propagation distances, all methods produce comparable results. At larger distances, the ISO 9613-2 and JRC-2012 method produce results that are significantly lower than the JRC-draft-2010 method.

4.1.12 Reflections from parallel walls

In order to observe the behaviour of the three methods in case of reflections from vertical walls, this test case considers a source in a 4m deep, 16m wide trench. Up to three reflections are taken into account, i.e. for each receiver four propagation paths are constructed: one direct path (diffracted by the edge at x=8m), one reflected by the opposite wall of the trench (at x=-8m), etc… Results are given separately for each order of reflections:

No reflections

direct (no reflections)							
method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	70.3	57.3	44.8	37.8	31.4	23.3
	Lp,H	-	-	-	-	-	-
	Leq	70.3	57.3	43.6	35.7	28.8	20.4
JRC-2012	Lp,F	75.7	57.2	43.2	33.8	27.4	20.6
	Lp,H	75.7	57.2	43.3	31.9	21.2	6.2
	Leq	75.7	57.2	43.2	32.9	25.3	17.7
JRC-draft-2010	Lp,F	75.6	54.6	39.1	31.1	25.7	20.9
	Lp,H	75.6	54.5	38.9	30.8	25.4	19.1
	Leq	75.6	54.5	39.0	30.9	25.6	20.1
1 reflection							
method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	64.4	56.4	46.1	39.4	33.6	-26.4
	Lp,H	-	-	-	-	-	-
	Leq	64.4	55.9	44.5	37.2	31.0	23.6
JRC-2012	Lp,F	64.6	60.5	45.9	36.9	29.6	22.7
	Lp,H	64.6	60.3	45.5	34.6	24.1	9.1
	Leq	64.6	60.4	45.7	35.9	27.7	19.9
JRC-draft-2010	Lp,F	64.0	58.4	41.4	31.8	24.2	21.3
	Lp,H	64.0	58.2	41.0	31.0	23.0	15.8
	Leq	64.0	58.3	41.2	31.4	23.6	19.4
2 reflections							
method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	56.9	48.2	43.1	37.7	32.3	24.7
	Lp,H	-	-	-	-	-	-
	Leq	56.9	47.0	41.2	35.4	29.6	21.9
JRC-2012	Lp,F	52.5	54.3	48.1	40.9	31.1	23.4
	Lp,H	52.5	54.3	44.5	34.4	24.3	9.2
	Leq	52.5	54.3	46.7	38.8	28.9	20.6

JRC-draft-2010	Lp,F	53.8	53.8	41.5	31.9	23.4	23.0
	Lp,H	53.9	53.8	40.7	30.3	20.9	12.6
	Leq	53.9	53.8	41.1	31.1	22.3	20.3
3 reflections							
method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	-	-	36.1	32.0	26.8	18.8
	Lp,H	-	-	-	-	-	-
	Leq	-	-	34.1	29.6	24.2	15.9
JRC-2012	Lp,F	44.6	44.9	44.5	37.2	34.9	27.3
	Lp,H	44.6	44.9	41.5	32.7	22.5	7.5
	Leq	44.6	44.9	43.2	35.5	32.2	24.3
JRC-draft-2010	Lp,F	45.6	46.9	39.6	30.6	22.6	23.3
	Lp,H	45.6	46.9	38.7	28.7	18.7	9.2
	Leq	45.6	46.9	39.2	29.8	21.1	20.4
sum direct + reflections							
method	level	10m	20m	50m	100m	200m	500m
ISO-9613-2	Lp,F	71.5	60.5	49.8	43.5	37.7	27.7
	Lp,H	-	-	-	-	-	-
	Leq	71.5	60.2	48.3	41.3	35.0	27.3
JRC-2012	Lp,F	76.0	62.9	51.8	44.0	37.7	30.2
	Lp,H	76.0	62.8	50.0	39.6	29.2	14.2
	Leq	76.0	62.8	51.0	42.3	35.3	27.3
JRC-draft-2010	Lp,F	75.9	61.0	46.5	37.4	30.2	28.3
	Lp,H	75.9	60.9	46.0	36.3	28.7	21.6
	Leq	75.9	61.0	46.3	36.9	29.5	26.1

For the total noise levels, differences larger than 4 dB(A) occur at all distances. It is not obvious to attribute these differences to a single phenomenon; i.e.it seems that in some cases, different approaches for calculating diffracted paths compensate for different ways of handling reflected paths…

4.2 Conformity checking

Options, either mandatory or used on a voluntary basis, control the behaviour of the calculation methods in a predictable way. A second series of test cases was designed specifically to verify the expected behaviour of the software in response to the settings of parameters and options.

4.2.1 File “road traffic + segment.xml”

This case uses the following extended features

- The importation of output file from source module: the elementary source is defined in an external file “CNOSSOS_Road_Output.xml”, as obtained on output of the road source module.
- The possibility to enter user defined temperature and humidity (set to 20°C, 60% RH). Because the calculation of air absorption is shared amongst the three propagation methods, the effect is the same in all three methods.

- The use of a specific meteorological model. This example forces meteorological effects to be accounted for by means of the ISO 9613-2 methodology (with C_0 = 3 dB), independently of selected calculation method.
- The possibility to specify user defined material properties: the ground covering the berm is set to "UserDefinedGround" with G= 1.0 and σ=50 kNs/m².
- Analytical integration of geometrical spread over a source line segment of finite length; i.e. the line source segment extends from (x=0, y=-10, z=0) till (x=0, y=10, z=0).

Results

	ISO-9613-2	JRC-2012	JRC-draft-2010
Lp,F	49.7	48.9	48.5
Lp,H	-	48.7	48.0
Leq	47.9	47.1	46.7

The geometrical spread can be calculated manually from $\Delta\alpha = 2.sin^{-1}(0.1),\ R = 100m$ and equals -38 dB as expected.

It can be noted that, in this case, setting C_0 = 3 dB leads to the prediction of long-time averaged noise levels that are lower than those under homogeneous propagation conditions.

The Harmonoise model predicts the additional effect of the user defined material which is more absorbing than the most absorbing ground (i.e. G = 1).

4.2.2 File "Road traffic + segment + height.xml"

Same as above, but the source height is manually set to 30 cm, thus overriding the standard value of 5 cm defined for the elementary source in the imported source file.

Results:

	ISO-9613-2	JRC-2012	JRC-draft-2010
Lp,F	50.3	49.2	49.2
Lp,H	-	48.9	48.6
Leq	48.6	47.5	47.5

As expected, all methods provide slightly higher noise levels due to a reduction of the ground effect.

4.2.3 File "point source + road traffic"

As above, the elementary source is defined in the external file "CNOSSOS_Road_Output.xml", but the elementary source is now associated with a point source in the propagation path.

As expected, all methods produce the same error:

```
ERROR: Sound power units do not match source geometry
No results available…
No output file generated
```

The software correctly checks that the sound power of the source is compatible with its geometrical shape. In this test case, the source module produces sound power per unit of length (relative to 10^{-12} W/m), whereas the point source geometry requires a characterisation in terms of total sound power (relative to 10^{-12} W).

4.2.4 File "point source.xml"

This case is identical to the previous one, but the option "CheckSoundPowerUnits" is now set to false. The output of the (line) source model is now (wrongly) interpreted as total sound power, or equivalently as the sound power produced by 1 meter of source length.

Results:

	ISO-9613-2	JRC-2012	JRC-draft-2010
Lp,F	36.7	35.9	35.5
Lp,H	-	35.7	35.0
Leq	34.9	34.1	33.7

4.2.5 Files "line source.xml" and "area source.xml"

As above, but the source is a now point source assimilated with a 10m long source line or with a 10m² area source.

Results

	ISO-9613-2	JRC-2012	JRC-draft-2010
line source			
Lp,F	46.7	45.9	45.5
Lp,H	-	45.7	45.0
Leq	44.9	44.1	43.7
area source			
Lp,F	46.7	45.9	45.5
Lp,H	-	45.7	45.0
Leq	44.9	44.1	43.7

As expected, the results are equal. Note that the size of the radiating object is integrated as part of the geometrical spread which is --51 dB(A) for the point source, -41 dB(A) for the line or area source.

4.2.6 Reflections

This test case simulates the reflection from a 1.5m high barrier on top of a 1 m high embankment on the opposite side of the source. The calculations are carried out for three different receiver heights:

File	Explanation
reflection-mid.xml	the receiver is located within the region of specular reflections
reflection-high.xml	the receiver is located within above the region of specular reflections
reflection-low.xml	the receiver is located within below the region of specular reflections

Results:

	ISO-9613-2	JRC-2012	JRC-draft-2010
reflection-mid			
Lp,F	23.1	50.7	51.2
Lp,H	-	51.2	50.2
Leq	22.6	51.0	50.7
reflection-high			
Lp,F	-	47.1	43.8
Lp,H	-	45.1	43.2
Leq	-	46.3	43.5
reflection-low			
Lp,F	-	47.5	48.4
Lp,H	-	47.5	48.4
Leq	-	47.5	48.4

ISO 9613-2 specifies that the specular reflection point must fall within the physical limits of the reflecting obstacle. Consequently, no path contribution is produced for the lower and higher receivers.

JRC-2012 and JRC-draft-2010 handle the lower receiver as a diffracting terrain edge in the unfolded propagation path. JRC-2012 handles the higher receiver by means of retro-diffraction; JRC-draft-2010 handles this configuration by means of Fresnel weighting.

Note that, even in case of specular reflections, the ISO 9613-2 method produces a much smaller result than the other methods because within each frequency band, potential contributions of reflected paths are taken into account (or not) based on the size of the Fresnel zone compared

to the height of the reflecting obstacle (e.g. in this example, only the 8000Hz band contributes to the partial noise level for this path).

Options can be used to increase the comparability of the three methods. As an example, the three files were modified and the CheckHeightLowerBound" and "CheckHeightUpperBound" options set to true.

Original files	Modified files
reflection-mid.xml	reflection-mid-comp.xml
reflection-high.xml	reflection-high-comp.xml
reflection-low.xml	reflection-low-comp.xml

As expected, all methods now produce results in case of a purely specular reflection but no path contributions for the higher / lower receivers.

For all three methods, the following error messages are displayed:

```
ERROR: Reflexion/diffraction below lower bound at position 2
No results available...
No output file generated
```

```
ERROR: Reflexion/diffraction above upper bound at position 2
No results available...
No output file generated
```

4.2.7 Lateral diffraction

File "lateral diffraction.xml" contains an example of a path diffracted by a single vertical edge.

Results

	ISO-9613-2	JRC-2012	JRC-draft-2010
Lp,F	43.0	43.0	-
Lp,H	-	42.9	-
Leq	41.8	41.8	-

As expected, we obtain no results for the JRC-draft-2010 method, which instead prints the following error message:

```
ERROR: Invalid path, no lateral diffractions allowed
No results available...
No output file generated
```

The file "lateral diffraction too-high.xml" is similar to the previous one but with a receiver higher up. Although this is not mentioned in the reference documents, we assume that lateral diffractions by vertical edges of finite height can be handled similarly to reflections:

- In the ISO 9613-2 method, it is assumed that the Keller diffraction point must fall within the bounds of the diffracting edge.
- In the JRC-2012 method, calculations are carried out as for a vertical edge of infinite length and diffraction and/or retro-diffraction in the unfolded propagation plane is used to take into account the finite size of the diffractions edge (similarly to the finite size of a reflection wall).

Results:

	ISO-9613-2	JRC-2012	JRC-draft-2010
Lp,F	-	35.7	-
Lp,H	-	35.7	-
Leq	-	35.7	-

File "diffraction not-convex.xml" illustrates a complex diffracted path. Currently, no method is able to handle such complex paths. The ISO and NMPB method print out the following error message:

```
ERROR: Invalid laterally diffracted path (not convex)
No results available...
No output file generated
```

4.2.8 Sound power adapter

This test case illustrates the influence of the source power measurement type in conjunction with the selected calculation method and the ground type. For this purpose, a point source is placed at 5 cm above a flat surface. The surface is either hard (class "H", G=0) or soft (class "D", G=1). The receiver is located at 10m distance, 4m above ground.

The different combinations are located in files

Lw undefined - hard.xml
Lw undefined - soft.xml
Lw freefield - hard.xml
Lw freefield - soft.xml
Lw hemispherical - hard.xml
Lw hemispherical - hard.xml

Where "freefield" refers to sound power measured under free field conditions (i.e. without any obstacles influencing the propagation of sound) and "hemispherical" to free field conditions above a hard surface (as specified e.g. in ISO 3745).

Results:

Lw measurement	undefined	free field	hemispherical
hard ground (G=0)			
ISO-9613-2	76.5	77.5	76.5
JRC-2012	76.5	77.5	76.4
JRC-draft-2010	77.8	77.8	76.8
soft ground (G=1)			
ISO-9613-2	72.9	73.6	72.9
JRC-2012	73.5	74.5	73.5
JRC-draft-2010	74.6	74.6	73.6

As can be seen from the results, "undefined" sound power measurement conditions correspond to the "default" interpretation for each method, i.e. ISO-9613-2 and JRC-2012 expecting sound power under hemispherical radiation conditions, JRC-draft-2010 expecting sound power under free field conditions.

As expected, the fact that the sound power is quantified as either free field or hemispherical improves the comparability of the different methods.

4.2.9 Inverse path definition

The calculation engine supports both direct (from source to receiver) and inverse (from receiver to source) propagation paths. It is assumed that reversing the propagation path will not influence the results.

The file "rec_to_source.xml" defines a propagation path with the receiver at the first position in the path construction. The file "source_to_rec.xml" contains the same propagation path but has the "ForceSourceToReceiver" option set to true. This implies that the calculation engine will, in its initial processing, reverse the path so that the source is located at the first position.

	ISO-9613-2	JRC-2012	JRC-draft-2010
rec_to_source.xml			
Lp,F	53.4	52.0	53.3
Lp,H	-	49.4	51.6
Leq	51.3	49.9	51.2
source_to_rec.xml			
Lp,F	53.4	52.0	53.3
Lp,H	-	49.4	51.6
Leq	51.3	49.9	51.2

Note that the NMPB method always sets the "ForceSourceToReceiver" option to true because the calculations are not reciprocal.

4.2.10 Simplifying terrain profiles

As first proposed by the Nord 2000 project, it may be an interesting feature to simplify the terrain profiles obtained from real GIS data before passing them on to the noise calculations.

The method proposed by the Nord2000 project has been implemented as an optional feature in the CNOSSOS-EU software. It can be enabled by setting the value of the "SimplifyPathGeometry" option to true.

The file "simplify terrain off.xml" contains a terrain with artificial roughness elements added. The file "simplify terrain on.xml" contains the same terrain profile but has the "SimplifyPathGeometry" option set to true.

	ISO-9613-2	JRC-2012	JRC-draft-2010
simplify terrain off.xml			
Lp,F	53.1	52.0	49.9
Lp,H	-	49.0	48.3
Leq	51.0	50.7	49.2
simplify terrain on.xml			
Lp,F	53.4	52.0	53.3
Lp,H	-	49.4	51.6
Leq	51.3	50.9	52.6

This example illustrates the fact that the ISO and NMPB methods are not sensitive to small details in the path geometry, probably due to the fact that all calculations are carried out over some implicit or explicitly constructed averaged ground plane. The Harmonoise model is more sensitive. This implies that accuracy and precision of the method are more strongly determined by the quality of the input data than in the other methods. On the other hand, simplification of the terrain profile reduces the calculation times by approx. 20% for the ISO and NMPB methods but divides it by a factor 8 in case of the Harmonoise method.

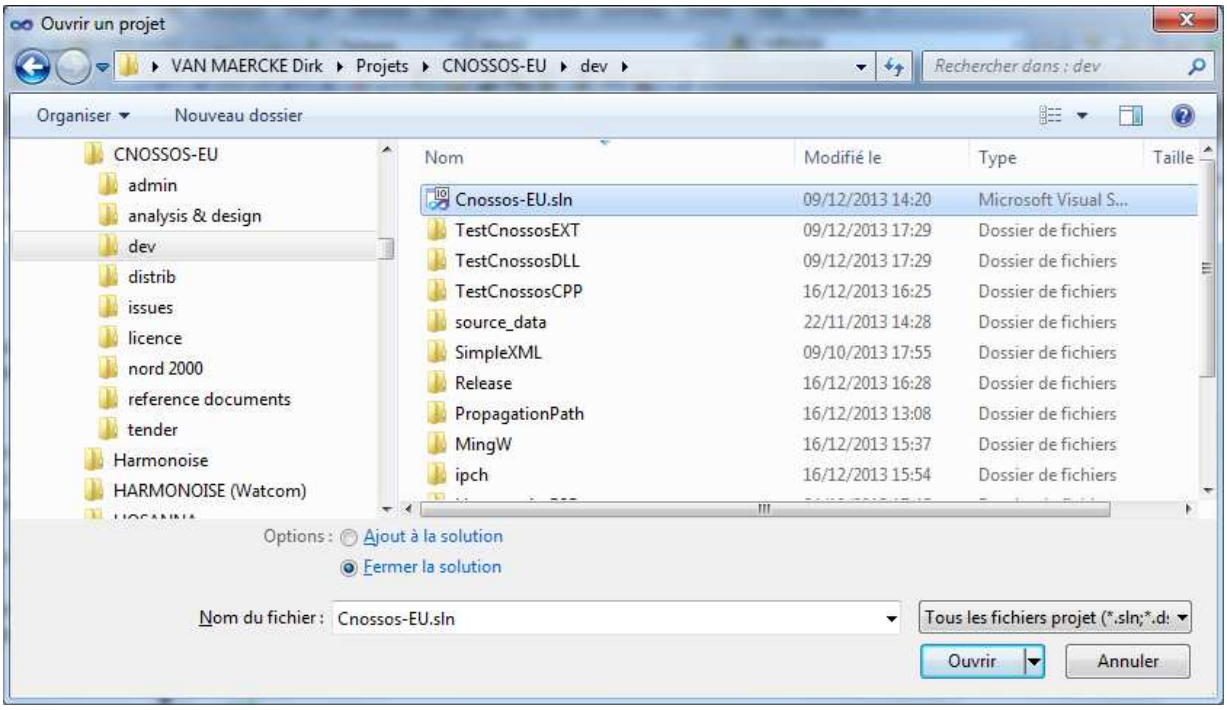
5 Building the software

5.1 Using Microsoft Visual Studio

The software modules and test applications were initially developed using Microsoft Visual Studio 2010, Professional Edition.

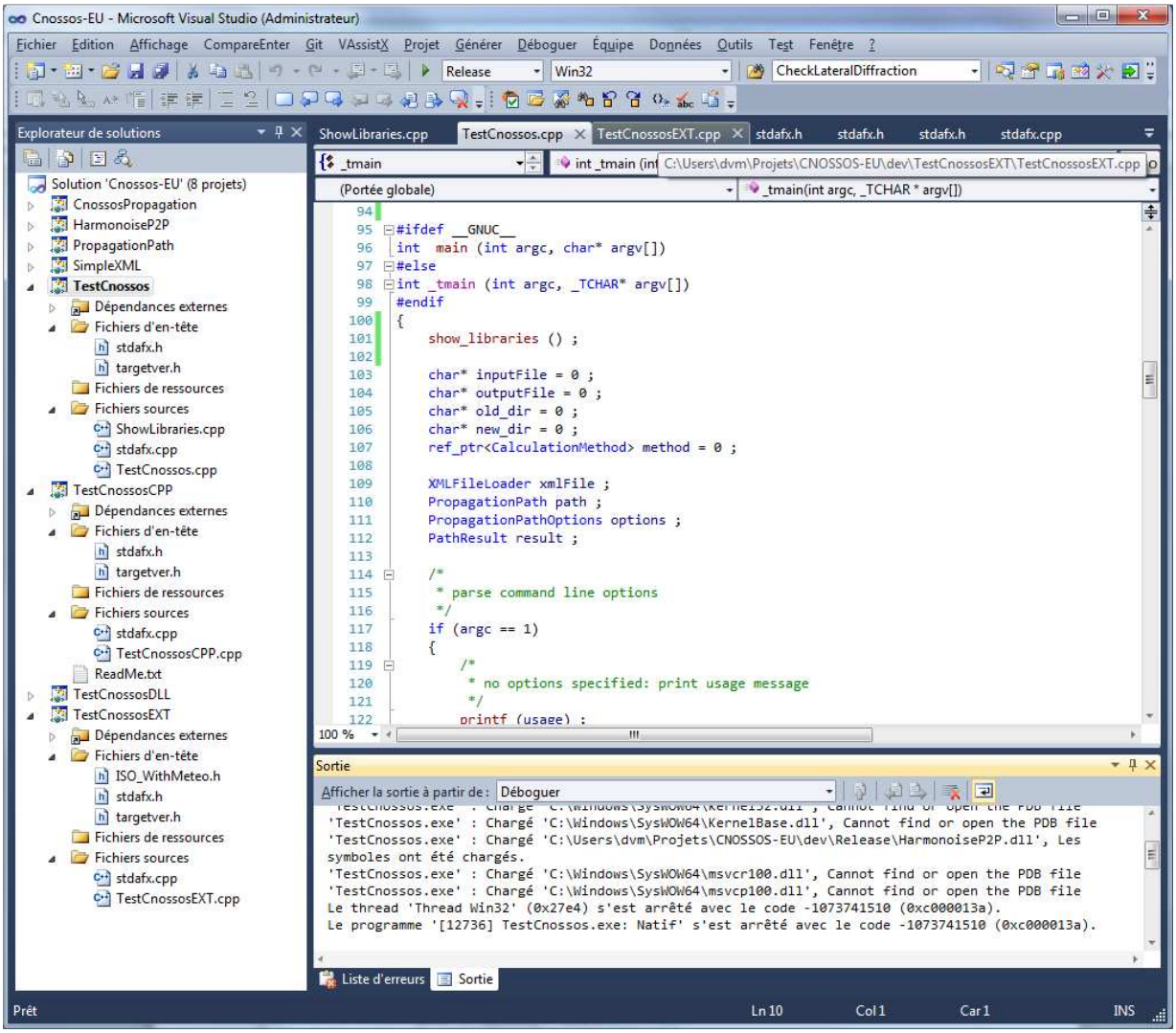
The project is compatible with the Visual C++ 2010 Express version which can be downloaded free of charge from http://www.visualstudio.com/downloads/download-visual-studio-vs.

After installing the appropriate version of Visual C++, open the "Cnossos-EU.sln" file, either from Windows Explorer or the "File/open/solution" command in Visual C++.



When loaded into Visual C++, the complete list of software modules is shown in the navigation panel (on the right).

The CNOSSOS-EU project (or "solution" in Microsoft terminology) contains eight modules (or "projects" in Microsoft terminology), four concern the libraries implementing the functionality of the different propagation modules, four concern different applications used for validation and demonstrations purposes.



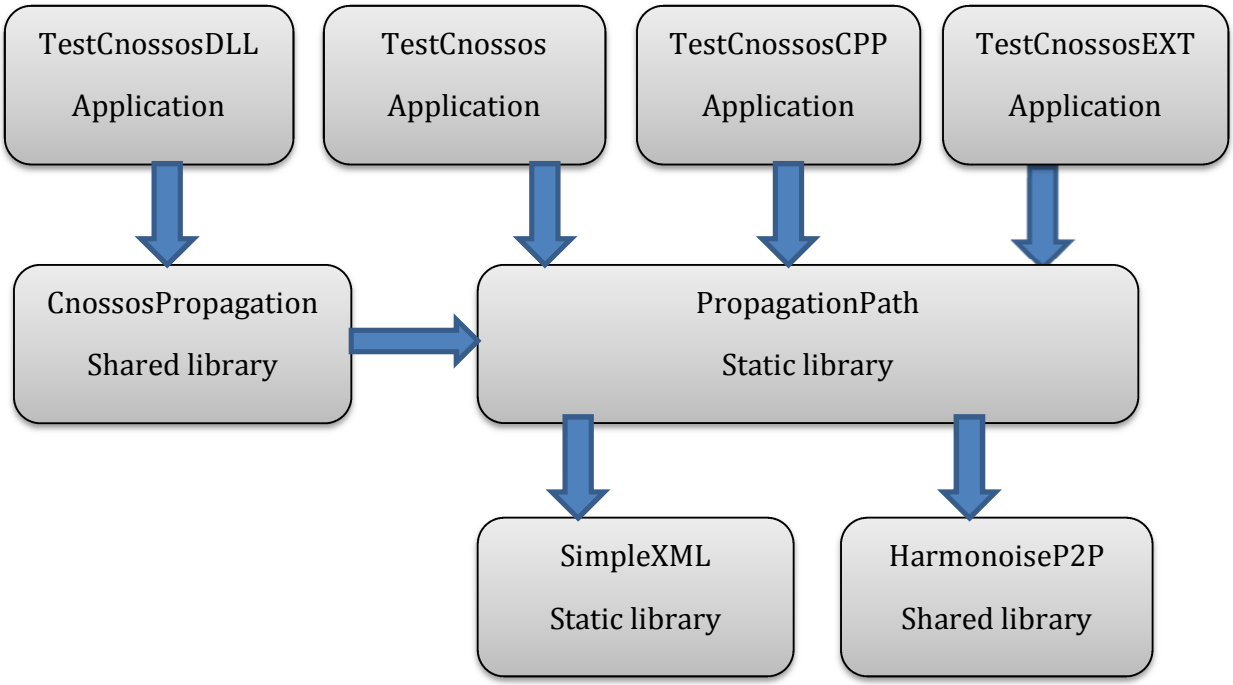
It is assumed that end-users wanting or needing to inspect and/or modify the code have sufficient knowledge of the C++ language and the Visual C++ development tools to do this without further assistance. In principle, the names of the modules, classes and functions are self-explaining and the code contains sufficient comments to clarify the purpose of each class, function, data member, enumerated constant,…

5.2 Modules and dependencies

The CNOSSOS-EU project contains the following modules:

Name	Contents	Type
SimpleXML	A light-weight XML file parser	static library
HarmonoiseP2P	Implementation of the Harmonoise propagation model in 2D	dynamic library
PropagationPath	Implementation of the three propagation models in 3D, including the shared functionalities for geometrical analysis of propagation paths and the management of material properties.	static library
CnossosPropagation	C-style programming interface to access the three propagation methods. Applications written in different programming languages can use this module in order to access the full functionality of the PropagationPath library, either by linking to the appropriate interface library (PropagationPath.lib) or by dynamically loading the dynamic library.	dynamic library
TestCnossosDLL	Test software illustrating the use of the CnossosPropagation dynamic library, using the C-style programming interface.	executable file
TestCnossos	Main application for testing and validation of the implementation of the three propagation modules. This application uses XML input and output files for the creation of user-defined propagation paths and to report calculation results. The use of this application is explained in full details in the previous sections of this document.	executable file
TestCnossosCPP	Test software illustrating the direct coupling of C++ application software with the core functionalities of the CNOSSOS-EU project. Using the C++ programming interface allows both for more efficient and more flexible code than when relying on the C-style wrapper interface.	executable file
TestCnossosEXT	Test software illustrating the advanced use of the PropagationPath library. Because the library exposes a C++ interface based on classes and virtual interfaces, it becomes easier for application software developers to integrate modifications to the core methods without breaking down existing application software building on the core functions.	executable file

The interdependency of the different modules is illustrated below:



5.3 The PropagationPath library

The core functions of the CNOSSOS-EU project, including the different propagation models can be found in the PropagationPath module. This module includes the following files:

Files	Functionality
CalculationMethod.h CalculationMethod.cpp	Abstract base class for all three propagation methods. Implements the shared framework and functions used in all three propagation methods.
ElementarySource.h	Data structures for storing sound powers associated with elementary sources.
ErrorMessage.h	Generic support for handling error conditions
Geometry3D.h	Basic operations on 2,3 and 4D vectors and positions
ISO-9613-2.h ISO-9613-2.cpp	Implements the ISO 9613-2 propagation model. The different components of the model are called through virtual functions defined in the abstract CalculationMethod class.
JRC-2012.h JRC-2012.cpp	Implements the NMPB-2008 propagation model as described in the JRC report, version 2012. The different components of the model are called through virtual functions defined in the abstract

	CalculationMethod class.
JRC-draft-2010.h JRC-draft-2010.cpp	Implements an interface on top of the Harmonoise propagation method as descrbed in the JRC reference report, version 2010. The different components of the model are called through virtual functions defined in the abstract CalculationMethod class. This class delegates calculations to the HarmonoiseP2P shared library.
Material.h Material.cpp	Implements the management of the shared material database and the automatic conversion functions for missing properties.
MeanPlane.h MeanPlane.cpp	Implements the calculation of the mean ground plane as defined in the JRC-2012 (NMPB) method.
MeteoCondition.h MeteoCondition.cpp	Implements the different meteorological models used in the evaluation of the long-term averaged noise levels. Meteorological averaging is based on ISO 9613-2 or NMPB-2008 and can be combined with any of the propagation methods.
PathParseXML.h PathParseXML.cpp	Implements the XML input file parser. Fills in the contents of a PropagationPath structure with data read from an external XML file.
PathResult.h PathResult.cpp	Data structure used to report the acoustical results returned by the calculation methods. Provides both global and detailed results, i.e. separate spectra for each component of the methods.
PropagationPath.h PropagationPath.cpp	Data structure used as input to the calculation methods, including the geometrical testing and processing of the propagation paths required starting the acoustical calculations.
ReferenceObject.h ReferenceObject.cpp	Memory management and automatic garbage collection.
SourceGeometry.h SourceGeometry.cpp	Data structures used to store the geometry of elementary point, line and area sources. Implements the evaluation of the attenuation due to geometrical spreading for each source type.
Spectrum.h Spectrum.cpp	Defines the frequency range of the calculation methods and data structures used for the storage of spectral data.
SystemClock.h SystemClock.cpp	Timer functions based on high-resolution system clock.
VerticalExt.h	Defines the vertical extensions associated with different positions in the propagation paths. Vertical extensions are used to code the source, the receiver, reflection walls, diffracting edges and thin barriers in the propagation path.

5.4 Calculation scheme

The main calculation scheme is implemented in the *CalculationMethod::doCalculation* method.

The calculation scheme first evaluates the different components of the noise calculation, then determines the partial noise levels associated with the path under favorable and homogeneous propagation conditions, and from these, determines the long-term averaged noise level. Finally, the results are converted to global dB(A) levels.

```
bool CalculationMethod::doCalculation (PropagationPath& path, PathResult& result)
{
        ...
        /*
         * evaluate the sound power of the source
         */
        result.Lw = options.ExcludeSoundPower ?
                    Spectrum(0.0) : getSoundPower (path) ;
        /*
         * convert sound power to dB(A) values if needed
         */
        result.dBA = getFrequencyWeighting (path) ;
        /*
         * evaluate the sound power adaptation of the source
         */
        result.delta_Lw = getSoundPowerAdaptation (path) ;
        /*
         * evaluate the geometrical spread
         */
        result.AttGeo = options.ExcludeGeometricalSpread ?
                        0.0 : getGeometricalSpread (path) ;
        /*
         * evaluate air absorption
         */
        result.AttAir = options.ExcludeGeometricalSpread ?
                        Spectrum (0.0) : getAirAbsorption (path) ;
        /*
         * evaluate attenuation due to absorption by reflecting obstacles
         */
        result.AttAbsMat = getAbsorption (path) ;
        /*
         * evaluate attenuation due to lateral diffraction
         */
        result.AttLatDif = getLateralDiffraction (path) ;
        /*
         * evaluate attenuation due to finite size of obstacles
         */
        result.AttSize = getFiniteSizeCorrection (path) ;
        /*
         * evaluate excess attenuation
         */
        result.AttF = getExcessAttenuation (path, true) ;
        result.AttH = getExcessAttenuation (path, false) ;
        /*
         * calculate levels
         */
        result.LpF = getNoiseLevel (result, true) ;
        result.LpH = getNoiseLevel (result, false) ;
        /*
```

```
            * estimate long-time averaged noise level
            */
           result.Leq = getNoiseLevel (path, result) ;
           /*
            * convert values to dB(A) values
            */
           result.LpF_dBA = getNoiseLevel (result.LpF) ;
           result.LpH_dBA = getNoiseLevel (result.LpH) ;
           result.Leq_dBA = getNoiseLevel (result.Leq) ;

           ...

           return true ;
    }
```

Note that most of the quantities manipulated in the calculation are of type "Spectrum", a convenience class that stores spectral values (i.e. values versus frequency) and redefines basic algebraic operations (such as sums and products) and mathematical functions (such as conversions between linear and logarithmic values) for the spectrum data type (see file Spectrum.h for details).

For example:

```
Spectrum A = …
Spectrum B = …
Spectrum C = A + B ;
```

Calls the "inline Spectrum operator+ (Spectrum const& s1, Spectrum const& s2)" operator and results in automatically generated code equivalent to

```
for (unsigned int i = 0 ; i < nbFreq ; ++i)
{
    C[i] = A[i] + B[i] ;
}
```

The evaluation of each component is implemented as a virtual function defined in the (abstract) CalculationMethod base class. Calculations that are shared amongst different methods are implemented in the base class, whereas functions that are specific to one of the three prediction methods are implemented in the derived classes ISO_9612_2, JRC2012 or JRCdraft2010.

The table below summarizes the location of the elementary calculations for each method.

Method	Description	Classes			
		CalculationMethod	ISO_9612_2	JRC2012	JRCdraft2010
getSoundPower	evaluate sound power of the	X			

	elementary source				
getFrequencyWeighting	apply dB(A) weighting if needed	X			
getSoundPowerAdaptation	convert between free-field and hemispherical sound power or vice-versa if needed	X			
getGeometricalSpread	evaluate attenuation due to finite source size and distance	X			
getAirAbsorption	evaluate attenuation due to air absorption	X			
getAbsoprtion	evaluate attenuation due to absorbing materials in case of reflected propagation paths	X			
getLateralDiffraction	evaluate attenuation due to diffraction around vertical edges		X	X	N.A.
getFiniteSizeCorrection	evaluate attenuation due to finite size of reflecting and/or diffracting obstacles		X	X	X
getExcessAttenuation	evaluate excess attenuation due to reflections on the ground and diffraction over obstacles, as a function of propagation conditions		X	X	X
getNoiseLevel	sum components and convert results to global dB(A) values	X			

5.5 IEEE compatible floating-point calculations

Note that the ISO method does not support calculations under homogeneous propagation conditions and that the corresponding noise level will be set to minus infinity ($-\infty$). The same holds in case the path cannot be evaluated by the selected propagation method or when a numerical error occurs during the calculation process. This occurrence of infinite values in the calculations is not problematic if the floating-point processor is set to "IEEE compatible" mode, which is the default for all modern C++ compilers. In this mode, no hardware exceptions are trapped and the processor correctly interprets operations on $\pm\infty$ values and mathematical functions return the expected limit values. For instance:

$$if\ (x = 0)\ then\ \ 10.\log(x) = -\infty$$

and

$$if\ (x = -\infty)\ then\ \ 10^{x/10} = 0$$

Therefore, no extra code is needed to check for infinite or undefined numerical values.

The symbolic constant "*negative_inf*" and "*positive_inf*" are defined in file Spectrum.h.

When using the software libraries with compilers and/or processors that do not support the IEEE extended range of floating point numbers, applications can disable the use of infinite values by calling the *CNOSSOS_EU_SetInfinityMode* function[1]. When set to false, the library will use (very) large positive and negative floating point numbers to represent infinite values.

5.6 Using the MingW/GCC compiler

As an alternative to the Microsoft Visual C++, the CNOSSOS-EU libraries and demo applications can be rebuilt using the MingW software. MingW is an implementation of the GNU/GCC compiler for Windows distributed under LGPL licensing conditions.

MingW can be downloaded free of charge from: http://sourceforge.net/projects/mingw/

The "dev/MingW" folder contains a sample distribution of the CNOSSOS-EU software built by means of the MingW/GCC compiler.

After downloading and installing the MingW compiler, open a command window, go to the "dev/MingW" directory and run the "Make.Cnossos.bat" script. The script will automatically build the different libraries and applications.

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW>dir
 Le volume dans le lecteur C n'a pas de nom.
 Le numéro de série du volume est 3208-A6F9

 Répertoire de C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW

16/12/2013  15:37    <REP>          .
16/12/2013  15:37    <REP>          ..
16/12/2013  16:02    <REP>          bin
16/12/2013  16:02    <REP>          lib
16/12/2013  16:02               261 Make.Cnossos.bat
16/12/2013  16:02    <REP>          obj
               1 fichier(s)              261 octets
               5 Rép(s)  168 702 287 872 octets libres

C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW>Make.Cnossos
```

If the build process succeeds, the script will run the main application:

```
C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW\bin>TestCnossos

Usage:

  TestCnossos [-w] [-m=<method>] [-i=<input file>] [-o] [-o=<output file>]
```

[1] As this function has been introduced in a very late stage of the project, it is not documented duly in the appendix B. Please refer to the *CnossosPropagation.h* file or to the *Spectrum.cpp* file for further information.

```
   .if -w is specified, the program will halt and wait for some user input
    before exiting, otherwise exit is automatic at the end of the calculations

   .method = ISO-9613-2, JRC-draft-2010 or JRC-2012; if not specified, the
    calculation will use the default method as found in the input file.

   .input file = the name of a valid XML file complying to the CNOSSOS-EU
    specifications.

   .output file = the name of the output file. If the file already exists,
    if will be replaced. If no filename is specified, output will be sent to
    the standard output stream for the active process.

    note that the output file will be searched and/or created relative to
    the folder containing the input file. If the file exists, it will be
    replaced or deleted (in case the program reports an error and no results
    can be produced).

    if -o is specified without a file name, the program will automatically
    generate one, based on the name of the input file name.

C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW\bin>cd ..
C:\Users\dvm\Projets\CNOSSOS-EU\dev\MingW>
```

6 Using the CnossosPropagation shared library

The CnossosPropagation.dll library provides a C-style application programming interface on top of the PropagationPath calculation engine. This API can be used from all programming languages that support linking or binding to C-style Windows DLL files. See the documentation of your programming language for details.

The full reference for the data structures, enumerated constants and functions used by the API is included in appendix C of this document. The TestCnossosDLL application illustrates the use of the API from a C-style main application (see file TestCnossosDLL.cpp).

6.1 Checking the library compatibility

In order to access the data structures, enumerated values and functions defined in the shared library, the application must include the appropriate header file:

```
#include "CnossosPropagation.h"
```

The demo software starts by checking the version of the API (as stored in the header file) against the version of the DLL (as returned by the GetVersion function).

```
/*
 * check version of API against DLL
 */
printf ("Testing CnossosPropagation.DLL library \n") ;
printf (".version API: %s\n", CNOSSOS_P2P_VERSION_API) ;
printf (".version DLL: %s\n", CNOSSOS_P2P_GetVersionDLL()) ;
```

6.2 Accessing material properties

Next, it illustrates how to access and print out material properties.

```
/*
 * illustrate access to material properties
 */
CNOSSOS_P2P_MATERIAL* mat = CNOSSOS_P2P_GetMaterial ("A4", false) ;
double G ;
double sigma ;
double alpha[8] ;
double freq[8] ;

CNOSSOS_P2P_GetFreq (freq) ;
CNOSSOS_P2P_GetGValue (mat,G) ;
CNOSSOS_P2P_GetSigma (mat, sigma) ;
CNOSSOS_P2P_GetAlpha (mat, alpha) ;

printf ("Material ID=""A4"", G=%.2f, sigma=%.0f\n", G, sigma) ;
print_spectrum ("freq:", "%4.0f", freq) ;
print_spectrum ("alpha:", "%4.2f", alpha) ;
```

Similar functions exist for creating new materials and/or modifying the properties of existing materials in the database. See the reference documentation in Appendix C.

6.3 Create the calculation engine

Next, the application creates an instance of the calculation engine, specifying one of the three calculation methods. Valid options are "JRC-draft-2010", "JRC-2012" and "ISO-9613-2".

```
/*
 * create a calculation method
 */
CNOSSOS_P2P_ENGINE* p2p = CNOSSOS_P2P_CreateEngine ("JRC-draft-2010") ;
printf ("Calculation method: %s, version %s\n",
         CNOSSOS_P2P_GetMethod (p2p), CNOSSOS_P2P_GetVersion (p2p)) ;
```

The pointer returned by this function is used in all subsequent calls to the library. Note that the functions for accessing or modifying the material properties do not use a pointer to a calculation engine because the material database is implemented as a unique instance (a singleton) shared by all calculation engines.

6.4 Create the propagation path

Next, the application starts a new propagation path and adds the source as its starting position.

```
/*
 * create a propagation path from the application
 */
CNOSSOS_P2P_ClearPath (p2p) ;
unsigned int nbPoints ;
/*
 * create source, i.e. a segment of a line source with known end-points
 */
CNOSSOS_P2P_POSITION pos ;
pos[0] = 0.0 ;
pos[1] = 0.0 ;
pos[2] = 0.0 ;
CNOSSOS_P2P_POSITION seg[2] ;
seg[0][0] = pos[0] ;
seg[0][1] = pos[1] - 10 ;
seg[0][2] = pos[2] ;
seg[1][0] = pos[0] ;
seg[1][1] = pos[1] + 10 ;
seg[1][2] = pos[2] ;
nbPoints = CNOSSOS_P2P_AddToPath (p2p, pos, CNOSSOS_P2P_GetMaterial("F"),
                                  CNOSSOS_P2P_CreateLineSource (0.5, seg)) ;
```

The fourth argument to the CNOSSOS_P2P_AddToPath function is a pointer to a vertical extension associated with the current position, in this case an elementary source line segment The footprint of the segment on the ground is running from (x=0, y=-10, z=0) till (x=0, y=10, z=0) and the equivalent point source is located at (x=0, y=0, z=0). The height of the source above local ground is 50 cm.

Note how the vertical extension is created "on the fly" and how the pointer returned by the CNOSSOS_P2P_CreateLineSource is not stored in any local variable. Because the library implements an integrated garbage collector, the memory allocated for the newly created vertical extension will be automatically destroyed as soon as the propagation path goes out of scope,

e.g. when the calculation engine is destroyed or when the path is cleared by means of the CNOSOS_P2P_ClearPath function.

The intersection of the propagation path with the ground is completed by inserting an impedance jump at (x = 2, y = 10) and the final point beneath the receiver at (x = 50, y = 10). The actual receiver position is located 2.5 m above local ground.

```
/*
 * hard surface beneath the source
 */
pos[0] = 10.0 ;
pos[1] =  2.0 ;
nbPoints = CNOSSOS_P2P_AddToPath (p2p, pos, CNOSSOS_P2P_GetMaterial("F")) ;
/*
 * soft ground till the receiver
 */
pos[0] = 50.0 ;
pos[1] = 10.0 ;
nbPoints = CNOSSOS_P2P_AddToPath (p2p, pos, CNOSSOS_P2P_GetMaterial("D"),
                                          CNOSSOS_P2P_CreateReceiver (2.5)) ;
```

6.5 Setting calculation options

The sound power of the source is specified by means of a separate call:

```
/*
 * define sound power associated with the source
 */
double Lw[] = { 90, 95, 100, 105, 105, 100, 95, 90 } ;
CNOSSOS_P2P_SetSoundPower (p2p, Lw) ;
```

The parameters for the meteorological model are set by first reading the current settings and then writing back the modified ones:

```
/*
 * setup meteorological weighting
 */
CNOSSOS_P2P_METEO meteo ;
CNOSSOS_P2P_GetMeteo (p2p, meteo) ;
meteo.model = CNOSSOS_P2P_METEO_JRC2012 ;
meteo.C0 = 3.0 ;
meteo.pFav = 0.50 ;
CNOSSOS_P2P_SetMeteo (p2p, meteo) ;
```

Similarly, the application can set or clear the different options influencing the behaviour and the compatibility of the calculation engine:

```
/*
 * setup options (optionally)
 */
CNOSSOS_P2P_OPTIONS options ;
CNOSSOS_P2P_GetOptions (p2p, options) ;
options.ExcludeSoundPower = false ;
options.ExcludeAirAbsorption = true ;
options.CheckSoundPowerUnits = = false ;
options.DisableLateralDiffractions = true ;
CNOSSOS_P2P_SetOptions (p2p, options) ;
```

6.6 Getting the results

Finally, the application requests the results for the current path and prints them to the console output. The request for any result will automatically launch the calculation on the current path. Similarly, modifying the path will delete any available results.

In case the GetResults function signals an error condition, the GetErrorMessage function is called in order to obtain a human readable explanation for it.

```
/*
 * read out the results (the first read will trigger the actual calculation)
 */
double fav[8] ;
double hom[8] ;
double leq[8] ;
unsigned int ok_fav
           = CNOSSOS_P2P_GetResult (p2p, CNOSSOS_P2P_RESULT_LP_FAV, fav) ;
unsigned int ok_hom
           = CNOSSOS_P2P_GetResult (p2p, CNOSSOS_P2P_RESULT_LP_HOM, hom) ;
unsigned int ok_leq
           = CNOSSOS_P2P_GetResult (p2p, CNOSSOS_P2P_RESULT_LP_AVG, leq) ;
/*
 * print out results
 */
if (ok_fav > 0 && ok_hom > 0 && ok_leq > 0)
{
       CNOSSOS_P2P_PrintPathResults (p2p) ;
       printf ("Results:\n") ;
       print_spectrum ("freq:", "%5.0f", freq) ;
       print_spectrum ("LpF:", "%5.1f", fav) ;
       print_spectrum ("LpH:", "%5.1f", hom) ;
       print_spectrum ("Leq:", "%5.1f", leq) ;
}
else
{
       printf ("ERROR: %s \n", CNOSSOS_P2P_GetErrorMessage(p2p)) ;
}
```

Notes:

- The CNOSSOS_P2P_GetResults function can be used to access individually each of the components of the noise calculation.
- The application must provide output buffers of sufficient size for the results to be returned. Some queries return a single value whereas others return spectral contents. In the present version of the software, all spectral results are encoded as arrays of 8 successive floating point numbers, corresponding to octave bands 63 till 8000 Hz.
- Alternatively, the application can call the CNOSSOS_P2P_GetFreq function with a NULL argument to obtain the number of values in each spectrum and then use this number to allocate the required memory dynamically.

6.7 Cleaning up

Note that a single instance of the calculation engine can be reused for the calculation of many different propagation paths. Do not forget to call the CNOSSOS_P2P_ClearPath function before

starting the definition of a new path. Calling the CNOSSOS_P2P_ClearPath function will invalidate any available results stored in the calculation engine. New results will be produced by the next call to the CNOSSOS_P2P_GetResults function.

When done, the application can free all memory associated with the calculation engine by calling the DeleteEngine function, passing in the pointer as returned by the CreateEngine function.

```
/*
 * cleanup
 */
CNOSSOS_P2P_DeleteEngine (p2p) ;
```

After calling the CNOSSOS_P2P_DeleteEngine function, the pointer is no longer valid and any reference to it (e.g. in any call to the library) may result in unexpected behaviour and/or memory corruption, possibly causing the immediate termination of the application.

7 Advanced programming

7.1 C++ programming interface

The CnossosPropagation library provides a lightweight C-style wrapper around the methods and data structures defined in the core PropagationPath library which has been written using the full power and functionality of the C++ language.

C++ programmers do not need to use the wrapper functions but may access the underlying classes and methods directly for higher computational efficiency and greater flexibility.

Direct linking of a C++ application with the PropagationPath library is illustrated in the TestCnossosCPP project.

As usual, the program starts by including the necessary header files.

```
#include "PropagationPath.h"
#include "Material.h"
#include "VerticalExt.h"
#include "CalculationMethod.h"
#include "PathResult.h"
```

In order to simplify the writing of the code, the application declares the name spaces used by the library.

```
using namespace CnossosEU ;
using namespace Geometry ;
```

Error handling is taken care of by means of C++ standard exception handling:

```
int _tmain(int argc, _TCHAR* argv[])
{
	try
	{
	    ...
	}
	catch (ErrorMessage& err)
	{
		err.print () ;
	}
}
```

The application creates a user-defined material and sets the appropriate properties:

```
/*
 * create a user-defined material and specify properties
 */
Material* mat = getMaterial ("UserDefinedGround", true) ;
mat->setG (0.6) ;
mat->setSigma (600) ;
```

The application creates a propagation path and fills in the positions and material attributes. Note that unspecified attributes are automatically initialized with default values; e.g. the material properties and vertical extensions associated with the path positions are all set to NULL unless specified differently in the application code.

```
/*
 * construct the geometry of the boundary beneath the path
 */
PropagationPath path ;
path.resize(4) ;
path[0].pos = Point3D ( 0.0,  0.0, 0.0) ;
path[1].pos = Point3D (10.0,  5.0, 0.0) ;
path[2].pos = Point3D (16.0,  8.0, 2.0) ;
path[3].pos = Point3D (50.0, 25.0, 2.0) ;
/*
 * specify materials associated with the boundary
 * note that materials are associated with the end points of the segments and
 * that specifying the material for the first position in the path is optional
 */
path[0].mat = path[1].mat = getMaterial ("F") ;
path[2].mat = path[3].mat = getMaterial ("UserDefinedGround") ;
```

The application then associates a short line source segment with the first position in the propagation path and a receiver with the last position.

```
/*
 * create the elementary source
 */
SourceExt* source = getSourceModel() ;
/*
 * add source geometry
 */
Point3D p1 (0.0, -2.5, 0.0) ;
Point3D p2 (0.0,  2.5, 0.0) ;
source->geo = new LineSegment (p1, p2) ;
/*
 * create the source at position 0
 */
path[0].ext = source ;
/*
 * create the receiver at position 3
 */
path[3].ext = new ReceiverExt (2.50) ;
```

The application then creates a calculation engine, sets up some calculation options and runs the paths through the engine, examines the results returned in the PathResult structure and saves the dB(A) level associated with this path in a local variable.

```
/*
 * create the propagation module
 */
CalculationMethod* method = getCalculationMethod ("JRC-draft-2010") ;
/*
 * set calculation options
 */
PropagationPathOptions options ;
options.meteo.C0 = 0 ;
options.meteo.model = MeteoCondition::ISO9613 ;
method->setOptions (options) ;
/*
 * run the calculation
 */
```

```
PathResult result ;
method->doCalculation (path, result) ;
/*
 * print details (in debug mode only)
 */
show_details (path, result) ;
/*
 * initial noise level in dB(A)
 */
double Leq_no_barrier = result.Leq_dBA ;
```

The application then modifies the path by adding a thin barrier at the third position in the path, i.e. at x = 16m and reruns the calculation (see the source code for details). It finally prints out the noise level associated with the path, with and without the barrier.

```
No barrier                                                    ----------------------
-------------------------------------------------
ID |       X |       Y |     Z |    G | Extension
-------------------------------------------------------------------------
00 |    0.00 |    0.00 |  0.00 | 0.30 | SRC h=0.00
01 |   10.00 |    5.00 |  0.00 | 0.30 |
02 |   16.00 |    8.00 |  2.00 | 0.60 |
03 |   50.00 |   25.00 |  2.00 | 0.60 | REC h=2.50
-------------------------------------------------------------------------
-------------------------------------------------------------------------
Freq(Hz) |    63 |   125 |   250 |   500 |  1000 |  2000 |  4000 |  8000
-------------------------------------------------------------------------
Lw       |  80.0 |  85.0 |  90.0 |  95.0 | 100.0 | 100.0 |  95.0 |  90.0
dB(A)    | -25.2 | -15.6 |  -8.4 |  -3.1 |   0.0 |   1.2 |   0.9 |  -2.4
deltaLw  |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0
AttGeo   | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0
AttAtm   |  -0.0 |  -0.0 |  -0.1 |  -0.1 |  -0.2 |  -0.5 |  -1.5 |  -5.3
AttRef   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttDif   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttSize  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
Att,F    |   5.7 |   4.5 |   1.2 |  -5.2 |  -5.2 |  -5.5 |  -4.7 |  -3.6
Att,H    |   5.6 |   4.3 |   1.0 |  -5.5 |  -5.7 |  -6.1 |  -5.6 |  -4.9
Lp,F     |  18.5 |  31.8 |  40.7 |  44.5 |  52.6 |  53.3 |  47.7 |  36.7
Lp,H     |  18.4 |  31.7 |  40.6 |  44.3 |  52.1 |  52.6 |  46.8 |  35.4
Leq      |  18.5 |  31.8 |  40.7 |  44.5 |  52.6 |  53.3 |  47.7 |  36.7
-------------------------------------------------------------------------
Lp,F     |  57.0 dB(A)
Lp,H     |  56.4 dB(A)
Leq      |  57.0 dB(A)
-------------------------------------------------------------------------
With barrier
-------------------------------------------------------------------------
ID |       X |       Y |     Z |    G | Extension
-------------------------------------------------------------------------
00 |    0.00 |    0.00 |  0.00 | 0.30 | SRC h=0.00
01 |   10.00 |    5.00 |  0.00 | 0.30 |
02 |   16.00 |    8.00 |  2.00 | 0.60 |
03 |   16.00 |    8.00 |  4.00 | 1.00 |
04 |   16.00 |    8.00 |  2.00 | 1.00 |
05 |   50.00 |   25.00 |  2.00 | 0.60 | REC h=2.50
-------------------------------------------------------------------------
-------------------------------------------------------------------------
Freq(Hz) |    63 |   125 |   250 |   500 |  1000 |  2000 |  4000 |  8000
-------------------------------------------------------------------------
```

```
Lw       |  80.0 |  85.0 |  90.0 |  95.0 | 100.0 | 100.0 |  95.0 |  90.0
dB(A)    | -25.2 | -15.6 |  -8.4 |  -3.1 |   0.0 |   1.2 |   0.9 |  -2.4
deltaLw  |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0 |  -3.0
AttGeo   | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0 | -39.0
AttAtm   |  -0.0 |  -0.0 |  -0.1 |  -0.1 |  -0.2 |  -0.5 |  -1.5 |  -5.3
AttRef   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttDif   |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
AttSize  |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0 |   0.0
Att,F    |   2.3 |  -2.6 | -11.8 | -15.0 | -12.7 | -16.3 | -19.0 | -20.9
Att,H    |   2.2 |  -2.7 | -11.9 | -15.3 | -12.9 | -16.6 | -19.2 | -21.1
Lp,F     |  15.1 |  24.8 |  27.8 |  34.8 |  45.1 |  42.4 |  33.4 |  19.4
Lp,H     |  15.0 |  24.7 |  27.7 |  34.5 |  44.8 |  42.1 |  33.2 |  19.2
Leq      |  15.1 |  24.8 |  27.8 |  34.8 |  45.1 |  42.4 |  33.4 |  19.4
-------------------------------------------------------------------------
Lp,F     |  47.5 dB(A)
Lp,H     |  47.2 dB(A)
Leq      |  47.5 dB(A)
-------------------------------------------------------------------------
Noise level without barrier = 57.0 dB(A)
Noise level with    barrier = 47.5 dB(A)
```

7.2 Using C++ inheritance

Application programmers can make use of the C++ inheritance to add new functionality without breaking existing code. Many of the classes used inside the PropagationPath Library have virtual functions that can be replaced with application-specific code. Note that major parts of the calculation scheme, as implemented in the NoiseCalculation class, are based on purely virtual functions that are redefined for each specific method. This mechanism facilitates the creation of derived methods that, while using most of the functionality of one of the existing methods, implement some modifications to others. In applications, such derived methods can then be used just like the official ones. This makes the evaluation of new methods and/or (candidate) modifications of existing methods (including comparison with the official versions) much easier than using any other programming technique.

This is illustrated in the TestCnossosEXT project.

The ISO method allows calculation for a single "moderately favorable" propagation condition. This value is then used to derive a long-time averaged noise level by means of a C_0 correction term. C_0 is defined by the end-user or by local authorities. In some countries/regions, a published method exist to link the C_0 correction term to local weather statistics. However, the C0 method has two major pitfalls: first, it is a constant for all propagation directions and secondly (and more important), it does not take into account the nature of the ground. It is a well-known fact that favorable propagation conditions mainly destroy ground effect (i.e. the destructive interference of a direct and a ground-reflected ray path that occurs under homogeneous propagation conditions). Therefore, the effects of meteorological conditions are generally much larger over soft ground than over hard ground and appropriate C_0 values should be defined ans used accordingly.

Alternatively, one might try to overcome the above mentioned difficulties by letting the ISO method make an estimate of the attenuation under both homogeneous and favorable propagation conditions. The frequency of occurrence of favorable and unfavorable propagation

conditions could then be used to evaluate the long-term averaged noise level, similarly to the French NMPB methods. Note that the frequency of occurrence of favorable propagation conditions can be evaluated in a well-defined way from historical recordings of meteorological data (e.g. from the ERA40 database) and this process can easily be automated for all European countries.

The TestCnossosEXT project introduces a new (user-defined) propagation method, derived from the official version of ISO-9613-2 (see file ISO_Withmeteo.h). The new method is implemented as a derived class using the ISO_9613_2 class as its base. The new class thus indirectly derives from the abstract CalculationMethod) and redefines only one method: the getExcessAttenuation method which is modified so that it returns excess attenuations under both homogeneous and favourable propagation conditions.

For this purpose, the new method calls the original getExcessAttenuation method (from the original ISO method) to get the excess attenuation under favourable conditions and then, if needed, applies some heuristic correction term to provide an estimate of the excess attenuation under homogeneous conditions (whereas the original method would return infinite attenuations when called with the favorable_condition flag set to false).

```
#include "ISO-9613-2.h"

using namespace CnossosEU ;

class IsoWithMeteo : public ISO_9613_2
{
public:

       inline const char* name (void) { return "IsoWithMeteo" ; }
       inline const char* version (void)  { return "0.001" ; }

       IsoWithMeteo(void) : ISO_9613_2() { }

       inline virtual Spectrum getExcessAttenuation (PropagationPath& path,
                                                     bool favorable_condition)
       {
           Spectrum att = ISO_9613_2::getExcessAttenuation (path, true) ;

           if (!favorable_condition)
           {
              ...
              att -= Cg ;
           }
           return att ;
    }
private:
    ...
} ;
```

The proposed heuristic correction is based on both the propagation distance and the nature of the ground through the averaged G value over the propagation path:

$$Att_H = Att_F - C_H . C_{DRS}$$

With:

$$C_H = \begin{cases} (1 + 5.G_{path}).\left[log\frac{d_P}{25}\right]^{1.5} & if\ d_P < 25m \\ 0 & if\ d_P < 25m \end{cases}$$

$$C_{DRS} = \begin{cases} 1 - 10\frac{h_S + h_R}{d_P} & if\ (h_S + h_R) < 10.d_P \\ 0 & if\ (h_S + h_{R)} > 10.d_P \end{cases}$$

The use of this modified propagation model is illustrated in the TestCnossosEXT main program.

The main program includes the necessary header file:

```
#include "ISO_WithMeteo.h"
```

It loads an XML path definition file from the hard disk and passes the path over to the modified calculation scheme, forcing the use the NMPB meteorological model (with 50% favorable conditions) for the determination of the long term averaged noise level:

```
/*
 * create the calculation engine
 */
ref_ptr<CalculationMethod> method ;
method = new MyExtension::IsoWithMeteo()
/*
 * force use of the NMPB meteorological model with 50% of favourable/homogeneous
 * propagation conditions
 */
options.meteo.model = MeteoCondition::JRC2012 ;
options.meteo.pFav = 0.50 ;
/*
 * set calculation options and process the propagation path
 */
method->setOptions (options) ;
method->doCalculation (path, result) ;
```

The results are shown below:

Note that, when the standard ISO propagation model is used with the NMPB meteorological weighting, the noise level under homogeneous propagation conditions will be set equal to minus infinity, i.e. assuming it is negligible small compared to the level under favorable conditions… Although this is technically possible, this combination may fail to provide accurate results.

7.3 Smart pointers

Many of the objects manipulated through the C++ application programming interface are created dynamically by means of operator "new". In order to avoid memory leaks, these objects should be deleted some later time, preferably before the end of the application. Because the access to such objects may be distributed amongst many classes and files, bookkeeping of

active and inactive objects rapidly becomes a nightmare. Worst of all, dynamically allocated objects may became inaccessible if the last reference (pointer) to them goes out of scope.

Intelligent pointers greatly help solving this kind of problems. No modern C++ program can do without them… Instead of building on some existing smart pointer library (e.g. from STL, boost, GT, OSG,…), the CNOSSOS-EU library implements its own light-weight mechanism supporting intelligent pointers and intrusive reference counting.

The mechanism is implemented in the file "ReferenceObject.h". It defines the ReferenceObject base class which implements the intrusive reference counting and the ref_ptr<T> template class as a smart pointer to instances of the ReferenceObject class. All abstract base classes in the PropagationPath project directly or indirectly derive from the ReferenceObject class and pointers to instances of such classes can be stored as smart pointers.

Internally, the PropagationPath library stores all pointers to dynamically created objects as smart pointers. This implies that all dynamically created objects will be deleted by the library when the last reference to them goes out of scope.

Typical use of dynamically created objects and smart pointers goes as following:

1. the application calls operator "new" to create a new instance of a given class type T,
2. the application passes the pointer (of type T*) as an argument to any call to the library,
3. the library converts the pointer into a smart pointer (of type ref_ptr<T>) and saves it in one of its internal data structures, this will increase the reference count inside the object by one.
4. the library now has permanent access to the dynamically created object and may use it in its calculations,
5. when the internal data structures are cleaned up, the smart pointers are reset to NULL,
6. when the reference count inside the dynamically created object falls to zero (i.e. it is no longer referenced by any smart pointer) it will be automatically destroyed.

Alternatively, applications may also store smart pointers to dynamically created objects, in which case the objects become permanent until the application releases the last reference to them.

Storing newly allocated objects in stack-based smart pointers ensures that unused objects (i.e. objects not passed on to and stored inside library) will be deleted before returning from a function call. Coherent use of smart pointers in combination with catch/try exception handling ensures that all temporarily created objects will effectively be deleted in case an error occurs inside the library.

Some care must be taken when returning pointers to dynamically allocated objects from user-defined functions (see file Referencebject.h for details).

7.4 Performances counters

Measuring the performance of an application in terms of actual processor time is not an obvious task, see e.g. http://demandtech.com/wp-content/uploads/2012/04/Measuring-Processor-Utilization-in-Windows.pdf. In view of this, one should clearly distinguish the elapsed time for an application to start, run and shutdown (which can be measured using a stopwatch) from the true time spent inside different stages of a calculation process.

For the purpose of monitoring the actual performances of the calculation processes, a highly accurate and precise timing functionality has been built inside the CNOSSOS-Eu library. The monitoring counts the number of calls to the point-to-point calculations and accumulates the total computation time spent inside the calculation functions (see file CalculationMethod.cpp).

```
bool CalculationMethod::doCalculation (PropagationPath& path,
                                       PathResult& result)
{
        SystemClock clock ;
        /*
         * setup local variables
         */
        initCalculation() ;
         .
         .
         .
        /*
         * cleanup local variables
         */
        exitCalculation() ;
        /*
         * update performance counters
         */
        nbCalls++ ;
        totalCPUTime += clock.get() ;

        return true ;
}
```

The values of these counters can be read by the application by means of:

```
class CalculationMethod : public System::ReferenceObject
{
public:
        ...
        /*
         * get performance counters
         */
        void getPerformanceCounter (unsigned int& _nbCalls, double& _cpuTime)
        {
                _nbCalls = nbCalls ;
                _cpuTime = totalCPUTime ;
        }
        ...
};
```

The reported timings are in seconds and have a typical resolution smaller than 10^{-6} sec.

Note that this function is equally available through the C-style programming interface (see file CnossosPropagation.h).

The use of the performance counter is illustrated in the TestCnossos.exe application.

7.5 Thread safety

In general, the calculations engine runs in a thread-safe way. Applications can create multiple instances of the calculation engine and run each instance in a separate thread.

Sharing static data structures, such as PropagationPath, PathOptions and PathResults between threads must be avoided unless the application can guarantee that no two threads will ever try to read and write the same structure simultaneously. This problem can easily be avoided if all input/output structures are created on the (thread-local) stack (as in the example above).

The automatic garbage collection takes care that dynamically created objects (such as vertical extensions for source, receiver, barriers, walls,…) in one thread will be deleted at the end of that thread ; i.e. when the PropagationPath containing such instances goes out of scope.

Care must be taken when writing to the he shared material database. Multi-threaded applications should carry out all write operations before starting the calculations, or create new Material records locally (in each thread) without inserting them in the shared database.

8 References

[1] Develop and implement Harmonised Noise Assessment Methods. Service contract ENV.C.3/SER/2012/0031.

[2] Develop and implement Harmonised Noise Assessment Methods. Inception report, Extrium, 15 March 2013.

[3] ISO 9613-2:1996. Acoustics – Attenuation of sound during propagation outdoors – Part 2: general method of calculation.

[4] JRC Reference Report on "Common Noise Assessment Methods in Europe (CNOSSOS-EU)", Draft version, May 2010, JRC (Ispra).

[5] JRC Reference Report on "Common Noise Assessment Methods in Europe (CNOSSOS-EU)", Final version, August 2012, EUR 25379 EN.

[6] Develop and implement Harmonised Noise Assessment Methods, Issues Log 2 associated with the technical review of JRC Reference Report, final version, August 2012.

[7] Develop and implement Harmonised Noise Assessment Methods, Issues Log 3 associated with the technical review of JRC Reference Report, draft version, May 2010.

[8] Develop and implement Harmonised Noise Assessment Methods, Issues Log 4 associated with the technical review of the ISO 1996-2:1996 standard.

[9] NF S 31-133, Bruit dans l'environnement - Calcul de niveaux sonores (février 2011).

[10] Besnard F. et.al.,"Prévision du bruit routier. Tome 2 – Méthode de calcul de la propagation du bruit incluant les effets météorologiques", Sétra (2009).

[11] Harmonoise WP3 – Engineering model for road and railway noise. Deliverable D18 of the Harmonoise project (IST-2000-28419).

[12] Van Maercke D, Defrance J, "Develoment of an Analytical Model for Outdoor Sound Propagation", Acta Acustica Vol.93(2007), 201-212.

[13] IMAGINE –WP1 – Specifications for GIS-NOISE databases. Deliverable 4 of the IMAGINE project (SSP1-CT-2003-503549).

[14] MingW, Minimalist GNU for Windows, http://www.mingw.org/

[15] Microsoft Download Center – Microsoft Visual Studio Express 2013 pour Windows Desktop http://www.microsoft.com/en-us/download/details.aspx?id=40787

[16] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985.

[17] IEEE Standard for Radix-Independent Floating-Point Arithmetic, ANSI/IEEE Std. 854-1987.

Appendix A: External modules and licensing conditions

The CNOSSOS-EU software has been built from different sources, including pre-existing modules from CSTB or third parties. As a general rule, the origin and licensing conditions of each module are indicated in the header section of each individual file.

For instance:

```
/*
 * ---------------------------------------------------------------------------
 * file:            Spectrum.h
 * version:         1.0
 * author:          dirk.van-maercke@cstb.fr
 * copyright:       see file licence.CSTB.txt
 * description:     ...
 *
 * ---------------------------------------------------------------------------
 */
```

The copyright notice refers to one of the following files, included in the distribution:

File	Origin
Licence.EXPAT.txt	Renamed version of the original "COPYING" file included as part of the EXPAT freeware distribution.
Licence.CSTB.txt	Pre-existing software owned by CSTB.
Licence.EU.txt	All software components specifically written under the provisions of service contract 070307/2012/633745/ENV.C3 and the related subcontract between Extrium and CSTB.

Library files distributed by Microsoft and/or automatically generated by the Visual Studio IDE are not specifically marked.

Each file indicates the specific property rights and associated licensing conditions for the use and exploitation of the software modules.

Expat is a freeware XML parser written by James Clark. The license grants permission to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, and to permit persons to whom the Software is furnished to do so. Expat is downloadable from http://sourceforge.net/projects/expat/.

The CNOSSOS-EU propagation software uses the EXPAT library for the parsing of XML-formatted input files (see file "PathParseXML.cpp"). Applications that link directly to the propagation modules by means of the C or C++ API's can be built and distributes without including the EXPAT modules.

Pre-existing software from CSTB is distributed under the European Union Public Licence v.1.1.

Pre-existing software from CSTB includes:

Files	Contents
\SimpleXML\SimpleXML.h \SimpleXML\SimpleXML.cpp	Lightweight XML-DOM buit on top of EXPAT.
\PropagationPath\Geometry3D.h	Elementary operations on 2,3 and 4D vectors.
\PropagationPath\ReferenceObject.h \PropagationPath\ReferenceObject.cpp	Low-level memory management and smart pointers.
\PropagationPath\Spectrum.h \PropagationPath\Spectrum.cpp	Data types and operations on spectral values.
\PropagationPath\SystemClock.h \PropagationPath\SystemClock.cpp	Timing functions using high-precision system clock.
\HarmonoiseP2P\PointToPoint.hpp \HarmonoiseP2P\PointToPointEx.hpp \HarmonoiseP2P\PointToPoint.cpp	The Harmonoise/Imagine 2D propagation model.

Appendix B: Cnossos-EU XML Reference Charts

The list below references the identifier (tag names) used in the context of the XML file used on input of the Cnossos-EU propagation module.

For each identifier are listed:

- the hierarchical context where the identifier may occur,
- the attributes of the tag,
- the contents of the XML entity ,
- for simple entities (i.e. end nodes of the XML graph), the type of the textual contents.

Complex types are nodes that contain other entities, i.e. have child nodes. E.g.

```
<complex_type>
    <child_node>
        …
    </child_node>
</complex_type>
```

Within a complex type, the order of child items is mandatory, i.e. must occur with the specified order. Multiplicity indicates the required (minimal / maximal) number of occurrences of child entities. Child entities may be optional (lower bound of the multiplicity equals zero) or mandatory (lower bound of the multiplicity equals one), allow a single or multiple instances (upper bound equal or greater than one). Some child entities may occur as often as needed ; their multiplicity is indicated as "any".

Simple types are entities that encode a single (numerical or textual) value but have no child nodes. E.g.

```
<simple_type> value  </simple_type>
```

Empty types are entities that have neither children nor contents. The only information associated with empty types is encoded as attributes. E.g.

```
<empty_type attribute="value" />
```

The list of identifiers below is in alphabetic order. Valid XML files must start with the <CNOSSOS-EU> root entity.

areaSource		
Context	CNOSSOS-EU → path → cp → ext → source → extGeometry	
Contents	Value	Multiplicity
area	numerical value	1
orientation	complex entity of type “pos”	0 or 1

barrier		
Context	CNOSSOS-EU → path → cp → ext	
Contents	Value	Multiplicity
h	numerical value	1
mat	empty type	0 or 1

CNOSSOS-EU		
Context	root node of the XML file	
Attributes	Value	Multiplicity
version	version number	0 or 1
Contents	Value	Multiplicity
method	complex entity	1
materials	complex entity	0 or 1
path	complex entity	1

cp		
Context	CNOSSOS-EU → path	
Contents	Value	Multiplicity
pos	complex entity	1
mat	empty type, if missing taken equal to the previous item in the list	0 or 1
ext	complex type	0 or 1

edge		
Context	CNOSSOS-EU → path → cp → ext	
Contents	Value	Multiplicity
h	numerical value	1

ext		
Context	CNOSSOS-EU → path → cp	
Contents	Value	Multiplicity
source	complex type	exactly one out of this list
receiver	complex type	
barrier	complex type	
wall	complex type	
edge	complex type	

extGeometry		
Context	CNOSSOS-EU → path → cp → ext → source	
Contents	Value	Multiplicity
pointSource	complex type	exactly one out of this list
lineSource	complex type	
areaSource	complex type	
lineSegment	complex type	

import		
Context	CNOSSOS-EU → path → cp → ext → source	
Contents	Value	Multiplicity
file	name of external file	1

lineSegment		
Context	CNOSSOS-EU → path → cp → ext → source → extGeometry	
Contents	Value	Multiplicity
posStart	complex entity of type “pos”	1
posEnd	complex entity of type “pos”	1
fixedAngle	numerical value	0 or 1

lineSource		
Context	CNOSSOS-EU → path → cp → ext → source → extGeometry	
Contents	Value	Multiplicity
length	numerical value	1
orientation	complex entity of type “pos”	0 or 1

Lw		
Context	CNOSSOS-EU → path → cp → ext → source	
Contents	list of 8 numerical values, corresponding to a spectrum in octave bands covering the range 63 – 8000 Hz	
Contents	Value	Multiplicity
sourceType	One of the following symbolic constants: “PointSource” “LineSource” “AreaSource”	0 or 1
measurementType	One of the following symbolic constants: “Undefined” “OmniDirectionnal” “HemiSpherical”	0 or 1
frequencyWeighting	One of the following symbolic constants: “LIN” or “dBA”	0 or 1

mat		
Context	CNOSSOS-EU → path → cp CNOSSOS-EU → path → cp → ext → barrier CNOSSOS-EU → path → cp → ext → wall	
Attribute	Value	Multiplicity
id	predefined or user-defined material identifier	1

material		
Context	CNOSSOS-EU → materials	
Attribute	Value	Multiplicity
id	unique material identifier, either predefined or user-defined	1
Contents	Value	Multiplicity
G	numerical value in the range 0 to 1	0 or 1
sigma	numerical value	0 or 1
alpha	list of 8 numerical values, corresponding to a spectrum in octave bands covering the range 63 – 8000 Hz	0 or 1

materials		
Context	CNOSSOS-EU	
Children	Value	Multiplicity
material	complex entity	any

method		
Context	CNOSSOS-EU	
Contents	Value	Multiplicity
select	empty type	1
options	complex entity	0 or 1
meteo	complex type	

meteo		
Context	CNOSSOS-EU → method → options	
Attribute	Value	Multiplicity
model	One of the following symbolic constants: “DEFAULT” “ISO-9613-2” “NMPB-2008” “JRC-2012”	1
Contents	Value	Multiplicity
temperature	numerical value in °C	0 or 1
humidity	numerical value, expressed as a percentage	0 or 1
pFav	numerical value, expressed as a percentage	0 or 1
C0	numerical value in dB	0 or 1

option		
Context	CNOSSOS-EU → method → options	
Attributes	Value	Multiplicity
id	One of the following symbolic constants: “CheckHorizontalAlignment", "CheckLateralDiffraction", "CheckHeightLowerBound", "CheckHeightUpperBound" "ForceSourceToReceiver", "DisableReflections", "DisableLateralDiffractions", "IgnoreComplexPaths", "ExcludeSoundPower", "ExcludeGeometricalSpread", "ExcludeAirAbsorption"	1
value	“true” or “false”	1

options		
Context	CNOSSOS-EU → method	
Contents	Value	Multiplicity
option	empty type	any

path		
Context	CNOSSOS-EU	
Contents	Value	Multiplicity
cp	complex entity	2 or more

pointSource		
Context	CNOSSOS-EU → path → cp → ext → source → extGeometry	
Contents	Value	Multiplicity
orientation	complex entity of type “pos”	0 or 1

pos		
Context	CNOSSOS-EU → path → cp CNOSSOS-EU → path → cp → ext → source → extGeometry → pointSource CNOSSOS-EU → path → cp → ext → source → extGeometry → lineSource CNOSSOS-EU → path → cp → ext → source → extGeometry → areaSource CNOSSOS-EU → path → cp → ext → source → extGeometry → lineSegment	
Contents	Value	Multiplicity
x	numerical value	0 or 1
y	numerical value	0 or 1
z	numerical value	0 or 1

select		
Context	CNOSSOS-EU → method	
Attributes	Value	Multiplicity
id	one of the following symbolic constants: “JRC-2012” “ISO-9613-2” “JRC-draft-2010”	1

source		
Context	CNOSSOS-EU → path → cp → ext	
Contents	Value	Multiplicity
import	complex type	0 or 1
h	numerical value	0 or 1*
Lw	simple type	0 or 1
extGeometry		

* if the <import> tag is missing, the <h> tag is mandatory

receiver		
Context	CNOSSOS-EU → path → cp → ext	
Contents	Value	Multiplicity
h	numerical value	1

wall		
Context	CNOSSOS-EU → path → cp → ext	
Contents	Value	Multiplicity
h	numerical value	1
mat	empty type	0 or 1

Appendix C: CnossosPropagation Programmer's Reference

Public functions defined in the CnossosPropagation shared library:

- **Create, configure and delete the calculation engine**

  CNOSSOS_P2P_CreateEngine
  CNOSSOS_P2P_DeleteEngine
  CNOSSOS_P2P_SelectMethod
  CNOSSOS_P2P_GetMethod
  CNOSSOS_P2P_GetVersion
  CNOSSOS_P2P_SetOptions
  CNOSSOS_P2P_GetOptions
  CNOSSOS_P2P_SetMeteo
  CNOSSOS_P2P_GetMeteo

- **Define the propagation path**

  CNOSSOS_P2P_ClearPath
  CNOSSOS_P2P_AddToPath
  CNOSSOS_P2P_SetSoundPower

- **Create vertical extensions**

  CNOSSOS_P2P_CreatePointSource
  CNOSSOS_P2P_CreateLineSource
  CNOSSOS_P2P_CreateReceiver
  CNOSSOS_P2P_CreateBarrier
  CNOSSOS_P2P_CreateVerticalWall
  CNOSSOS_P2P_CreateVerticalEdge

- **Get the acoustical results**

  CNOSSOS_P2P_GetResult
  CNOSSOS_P2P_PrintPathData
  CNOSSOS_P2P_PrintPathResults

- **Manage material properties**

  CNOSSOS_P2P_GetMaterial
  CNOSSOS_P2P_GetGValue
  CNOSSOS_P2P_SetGValue
  CNOSSOS_P2P_GetSigma
  CNOSSOS_P2P_SetSigma
  CNOSSOS_P2P_GetAlpha
  CNOSSOS_P2P_SetAlpha

- **Utility functions**

  CNOSSOS_P2P_GetVersionDLL
  CNOSSOS_P2P_GetErrorMessage
  CNOSSOS_P2P_ProcessPathFile

Function Documentation

CNOSSOS_P2P_AddToPath

unsigned int CNOSSOS_P2P_AddToPath (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_POSITION const& _pos_, CNOSSOS_P2P_MATERIAL *_mat_, CNOSSOS_P2P_EXTENSION *_ext_ = 0)

Add a new control point to the propagation path.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine
pos	Position of the control point
mat	Transparent pointer to a material as returned by a previous call to **CNOSSOS_P2P_GetMaterial**
ext	Transparent pointer to a vertical extension record as returned by a previous call to **CNOSSOS_P2P_CreatePointSource**, **CNOSSOS_P2P_CreateLineSource**, **CNOSSOS_P2P_CreateReceiver**, **CNOSSOS_P2P_CreateBarrier**, **CNOSSOS_P2P_CreateVerticalWall** or **CNOSSOS_P2P_CreateVerticalEdge**. For intermediate boundary positions without vertical extensions, this argument is set to NULL.

Returns:

The number of positions in the propagation path after appending the specified control point

Note:

A control point is part of the boundary separating the solid 2.5D model from the air above. Control points can be located on the ground or on top of man-made obstacles. Altitude of control points is measured against a geo-spatial reference coordinate system. Vertical extensions encode information related to vertical objects extending above the control point's position. The height of a vertical extension is measured relative to the altitude of the control point.
A valid path should have exactly one source and one receiver vertical extension at its extreme points. Paths may start at the source and end with a receiver or vice-versa.

CNOSSOS_P2P_ClearPath

bool CNOSSOS_P2P_ClearPath (CNOSSOS_P2P_ENGINE *_p2p_)

Clear the propagation path data.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine

Returns:

true if the call succeeded, false otherwise

Note:

A propagation path is created by successive calls to **CNOSSOS_P2P_AddToPath**. A call to **CNOSSOS_P2P_GetResult** will close the path and trigger the acoustical calculations. In order to restart the calculation for a different path, applications must call the **CNOSSOS_P2P_ClearPath** function before constructing a new one.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_CreateBarrier

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreateBarrier (double const& *h*, CNOSSOS_P2P_MATERIAL **mat* = `0`)

Create a vertical extension for a thin barrier crossed by the propagation plane.

Parameters:

h	height of the barrier above the boundary profile
mat	Pointer to the material covering the barrier, as returned by a call to **CNOSSOS_P2P_GetMaterial**

Returns:

A transparent pointer to a vertical extension record

CNOSSOS_P2P_CreateEngine

CNOSSOS_P2P_ENGINE* CNOSSOS_P2P_CreateEngine (const char **name* = `0`)

Create a new calculation engine.

Parameters:

name	If specified, initializes the engine for the specified calculation method

See also:

CNOSSOS_P2P_SelectMethod

Returns:

An opaque pointer to the newly created calculation engine

CNOSSOS_P2P_CreateLineSource

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreateLineSource (double const& *h*, CNOSSOS_P2P_POSITION const *segment*[2], double const& *fixedAngle* = `0.0`)

Create a vertical extension representing a line source segment.

Parameters:

h	height of the source above the boundary profile
segment	end-points of the source line segment as projected on the ground
fixedAngle	explicit angle of view of the segment as seen from the receiver.

Returns:

A transparent pointer to a vertical extension record

Note:

Setting a fixed angle is necessary in case (inverse) ray-tracing is used for constructing the propagation path as each ray is considered representative of an implicit sector of propagation with fixed opening angle and having the propagation path as its bisector. If the fixed angle argument is set to zero, the calculation will explicitly calculate the angle of view of the segment as seen from the source, which is typically the case when using beam-tracing or image-source techniques for constructing the propagation paths.

CNOSSOS_P2P_CreatePointSource

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreatePointSource (double const &*h*)

Create a vertical extension representing a point source.

Parameters:

h	height of the point source above the boundary profile

Returns:

A transparent pointer to a vertical extension record

CNOSSOS_P2P_CreateReceiver

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreateReceiver (double const &*h*)

Create a vertical extension representing a receiver point.

Parameters:

h	height of the receiver above the boundary profile

Returns:

A transparent pointer to a vertical extension record

CNOSSOS_P2P_CreateVerticalEdge

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreateVerticalEdge (double const &*h*)

Create an extension for a vertical edge causing lateral diffraction of the propagation path.

Parameters:

h	height of the diffracting edge above the boundary profile

Returns:

A transparent pointer to a vertical extension record

CNOSSOS_P2P_CreateVerticalWall

CNOSSOS_P2P_EXTENSION* CNOSSOS_P2P_CreateVerticalWall (double const &*h*, CNOSSOS_P2P_MATERIAL **mat* = 0)

Create an extension for a vertical wall acting as a reflector for the propagation path.

Parameters:

h	height of the reflecting wall above the boundary profile
mat	Pointer to the material covering the reflecting surface, as returned by a call to **CNOSSOS_P2P_GetMaterial**

Returns:

A transparent pointer to a vertical extension record

CNOSSOS_P2P_DeleteEngine

bool CNOSSOS_P2P_DeleteEngine (CNOSSOS_P2P_ENGINE **p2p*)

Delete a calculation engine created by a call to **CNOSSOS_P2P_CreateEngine**.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetAlpha

bool CNOSSOS_P2P_GetAlpha (CNOSSOS_P2P_MATERIAL **mat*, double **alpha*)

Get the spectral absorption factor for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
alpha	Array containing the values of the absorption factor spectrum

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetErrorMessage

const char* CNOSSOS_P2P_GetErrorMessage (CNOSSOS_P2P_ENGINE *_p2p_)

Get a description of the last error that occurred in the acoustical calculation.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.

Returns:

Pointer to a string containing the error message.

CNOSSOS_P2P_GetFreq

unsigned int CNOSSOS_P2P_GetFreq (double *_freq_)

Get the frequency range for the spectral data.

Parameters:

freq	Array in which to return the frequency bands associated with spectral information passed to and from the shared library. If a NULL pointer is specified, the function returns the number of values that would have been written on output but does not transfer any actual values.

Returns:

The number of frequency bands.

Note:

It is up to the caller to pass in an array of sufficient size. Applications can determine the minimal size of this array by first calling the function with a NULL argument.

CNOSSOS_P2P_GetGValue

bool CNOSSOS_P2P_GetGValue (CNOSSOS_P2P_MATERIAL *_mat_, double &_G_)

Get the ground factor for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
G	New value of the ground factor

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetMaterial

CNOSSOS_P2P_MATERIAL* CNOSSOS_P2P_GetMaterial (const char *_id_, bool _create_if_needed_ = `false`)

Get material pointer.

Parameters:

id	Unique textual identifier for the material
create_if_needed	If a material with the given name does not exist and this flag is set to true, the library will create a new material; otherwise this function returns a NULL pointer.

Returns:

An opaque pointer to the material or NULL if no material with the given name exists and the create_if_needed flag is set to false.

CNOSSOS_P2P_GetMeteo

bool CNOSSOS_P2P_GetMeteo (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_METEO &_meteo_)

Get the meteorological condition parameters.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
meteo	Data structure used to return the current settings of the meteorological parameters

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetMethod

const char* CNOSSOS_P2P_GetMethod (CNOSSOS_P2P_ENGINE *_p2p_)

Get the name of the currently selected propagation path calculator.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine

Returns:

Pointer to a string containing the name of the method

CNOSSOS_P2P_GetOptions

bool CNOSSOS_P2P_GetOptions (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_OPTIONS &_options_)

Get the current settings of the calculation options.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
options	Data structure used to return the current settings.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetPerformanceCounters

bool CNOSSOS_P2P_GetPerformanceCounters (CNOSSOS_P2P_ENGINE *_p2p_, unsigned int & _nbCalls_, double & _cpuTime_)

Get the performance counters for the specified instance of the calculation engine.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
nbCalls	Number of calls to this instance of the path calculator
cpuTime	Total elapsed time spent inside the acoustical path calculations

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetResult

unsigned int CNOSSOS_P2P_GetResult (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_RESULT_index_, double *_result_)

Get the acoustical results for the current propagation path.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
index	Select the type of results to be returned
result	Array in which to return the acoustical results. If a NULL pointer is specified, the function returns the number of values that would have been written on output but does not transfer any actual values.

Returns:

The number of values written (or not) to the array of results or zero in case an error occurred in the calculation.

Note:

It is up to the caller to pass in an array of sufficient size. Applications can determine the minimal size of this array by first calling the function with a NULL argument.

CNOSSOS_P2P_GetSigma

bool CNOSSOS_P2P_GetSigma (CNOSSOS_P2P_MATERIAL *_mat_, double &_sigma_)

Get the flow resistivity for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
sigma	Value of the flow resistivity (in kPa.s/m²)

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_GetVersion

const char* CNOSSOS_P2P_GetVersion (CNOSSOS_P2P_ENGINE *_p2p_)

Get the version of the currently selected propagation path calculator.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine

Returns:

Pointer to a string containing the current version of the method

CNOSSOS_P2P_GetVersionDLL

char* CNOSSOS_P2P_GetVersionDLL (void)

Get the current version of the CnossosPropagation shared library.

Returns:

String encoded version of the shared library.

CNOSSOS_P2P_PrintPathData

bool CNOSSOS_P2P_PrintPathData (CNOSSOS_P2P_ENGINE *_p2p_)

Print the current propagation path to the application's default output device.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_PrintPahtResults

bool CNOSSOS_P2P_PrintPathResults (CNOSSOS_P2P_ENGINE *_p2p_)

Print an overview of the acoustical results to the application's output device.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_ProcessPathFile

bool CNOSSOS_P2P_ProcessPathFile (CNOSSOS_P2P_ENGINE *_p2p_, const char *_inputFile_, const char *_outputFile_ = `0`)

Process a XML input file and optionally produce an XML output file.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
inputFile	Name of an XML input file
outputFile	Name of an (optional) XML output file

Returns:

true if the call succeeded, false otherwise

Note:

The name of the input file can include a full path specification or be a path name relative to the current working directory. The name of the output file can be either a full path specification or a path name relative to the location of the input file. Even if no output file is generated, the application can use the **CNOSSOS_P2P_GetResult** function to access the acoustical results associated with the path defined in the input file.

CNOSSOS_P2P_SelectMethod

bool CNOSSOS_P2P_SelectMethod (CNOSSOS_P2P_ENGINE *_p2p_, const char *_name_)

select the point to point method

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine
name	Textual identifier of the calculation method. Valid methods include "ISO-9613-2", "JRC-DRAFT-2010" and "JRC-2012"

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetAlpha

bool CNOSSOS_P2P_SetAlpha (CNOSSOS_P2P_MATERIAL *_mat_, double *_alpha_)

Set the flow resistivity for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
sigma	New value of the flow resistivity (in kPa.s/m²)

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetGValue

bool CNOSSOS_P2P_SetGValue (CNOSSOS_P2P_MATERIAL *_mat_, double const &_G_)

Set the ground factor for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
G	New value of the ground factor

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetMeteo

bool CNOSSOS_P2P_SetMeteo (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_METEO const &_meteo_)

Set the meteorological condition parameters.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
meteo	Data structure containing the new values of the meteorological parameters.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetOptions

bool CNOSSOS_P2P_SetOptions (CNOSSOS_P2P_ENGINE *_p2p_, CNOSSOS_P2P_OPTIONS const &_options_)

Set the current settings of the calculation options.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
options	Data structure containing the new values of the options.

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetSigma

bool CNOSSOS_P2P_SetSigma (CNOSSOS_P2P_MATERIAL *_mat_, double const &_sigma_)

Set the flow resistivity for a given material.

Parameters:

mat	An opaque pointer returned by a call to **CNOSSOS_P2P_GetMaterial**
sigma	New value of the flow resistivity (in kPa.s/m²)

Returns:

true if the call succeeded, false otherwise

CNOSSOS_P2P_SetSoundPower

bool CNOSSOS_P2P_SetSoundPower (CNOSSOS_P2P_ENGINE *_p2p_, double const *_Lw_, CNOSSOS_P2P_SPECTRUM_WEIGHTING_spectrumWeighting_ = `CNOSSOS_P2P_SPECTRUM_LIN`, CNOSSOS_P2P_LW_MEASUREMENT_TYPE_measurementType_ = `CNOSSOS_P2P_LW_UNDEFINED`)

Set the sound power associated with the source.

Parameters:

p2p	An opaque pointer returned by a previous call to CNOSSOS_P2P_CreateEngine.
Lw	Pointer to an array containing the sound power spectrum.
spectrumWeighting	Weighting of the sound power spectrum
measurementType	Measurement conditions for the sound power spectrum

Returns:

true if the call succeeded, false otherwise

Data Type Documentation

CNOSSOS_P2P_ENGINE

struct CNOSSOS_P2P_ENGINE

Opaque pointer to hidden class structure containing the for path calculator.

CNOSSOS_P2P_EXTENSION

struct CNOSSOS_P2P_EXTENSION

Opaque pointer to vertical extension record.

CNOSSOS_P2P_LW_MEASUREMENT_TYE

enum CNOSSOS_P2P_LW_MEASUREMENT_TYPE

Sound power measurement conditions.

Enumerator:

CNOSSOS_P2P_LW_UNDEFINED Measurement conditions undefined

CNOSSOS_P2P_LW_FREEFIELD Sound power measured under free field conditions

CNOSSOS_P2P_LW_HEMISPHERICAL Sound power measured under hemispherical conditions

CNOSSOS_P2P_MATERIAL

struct CNOSSOS_P2P_MATERIAL

Opaque pointer to material property record.

CNOSSOS_P2P_METEO

struct CNOSSOS_P2P_METEO

Meteorological input data.

Public Members

double C0

C0 correction term as defined in ISO 9613-2

double humidity

Air relative humidity (expressed as %)

CNOSSOS_P2P_METEO_MODEL model

Choice of meteorological model

double pFav

Frequency of occurrence of favorable propagation conditions

double temperature

Air temperature

CNOSSOS_P2P_METEO_MODEL

enum CNOSSOS_P2P_METEO_MODEL

Choice of meteorological averaging.

Enumerator:

CNOSSOS_P2P_METEO_DEFAULT Use default model corresponding to the selected calculation method

CNOSSOS_P2P_METEO_ISO9613 Use the C0 correction as defined in the ISO 9613-2 standard

CNOSSOS_P2P_METEO_JRC2012 Use the frequency of occurence of favorable propagation conditions

CNOSSOS_P2P_OPTIONS

struct CNOSSOS_P2P_METEO

Options modifying the behavior of the calculation engine.

Public Members

bool CheckHeightLowerBound

check height of ray path with respect to upper limits of vertical obstacles

bool CheckHeightUpperBound

check height of ray path with respect to lower limits of vertical obstacles

bool CheckHorizontalAlignment

check validity of paths in the horizontal plane

bool CheckLateralDiffraction

check validity of laterally diffracted paths

bool CheckSoundPowerUnits

check conformity of sound power units and source geometry

bool CheckSourceSegment

check whether source line segments contains actual source position

bool DisableLateralDiffractions

disable lateral diffractions

bool DisableReflections

disable reflections

bool ExcludeAirAbsorption

include atmospheric absorption

bool ExcludeGeometricalSpread

include geometrical spread

bool ExcludeSoundPower

include sound power

bool ForceSourceToReceiver

reverse the path in case it is ordered from Receiver to Source

bool IgnoreComplexPaths

disable reflections and vertical diffraction for laterally diffracted paths

bool SimplifyPathGeometry

remove non representative terrain points

CNOSSOS_P2P_POSITION

typedef double CNOSSOS_P2P_POSITION[3]

Encoding of positions using 3 dimensional (X,Y,Z) coordinates.

CNOSSOS_P2P_RESULT

enum CNOSSOS_P2P_RESULT

Select the result to be returned by the call to **CNOSSOS_P2P_GetResult**.

Enumerator:

CNOSSOS_P2P_RESULT_LP_AVG_dBA

global, dB(A) weighted, long term averaged noise level

CNOSSOS_P2P_RESULT_LP_FAV_dBA

global, dB(A) weighted, noise level under favorable propagation conditions

CNOSSOS_P2P_RESULT_LP_HOM_dBA

global, dB(A) weighted, noise level under homogeneous propagation conditions

CNOSSOS_P2P_RESULT_LP_AVG

long-time averaged noise level spectrum

CNOSSOS_P2P_RESULT_LP_FAV

noise level spectrum under favorable propagation conditions

CNOSSOS_P2P_RESULT_LP_HOM

noise level spectrum under homogeneous propagation conditions

CNOSSOS_P2P_RESULT_ATT_FAV

excess attenuation spectrum under favorable propagation conditions

CNOSSOS_P2P_RESULT_ATT_HOM

excess attenuation spectrum under homogeneous propagation conditions

CNOSSOS_P2P_RESULT_LW_SOURCE

sound power spectrum

CNOSSOS_P2P_RESULT_DELTA_LW

sound power conversion, free field versus hemispherical radiation conditions

CNOSSOS_P2P_RESULT_ATT_GEO

attenuation due to geometrical spreading

CNOSSOS_P2P_RESULT_ATT_ATM

attenuation due to atmospheric absorption

CNOSSOS_P2P_RESULT_ATT_REF

attenuation due to reflections by vertical obstacles

CNOSSOS_P2P_RESULT_ATT_DIF

attenuation due to lateral diffraction by vertical edges

CNOSSOS_P2P_RESULT_ATT_SIZE

attenuation due to finite size of vertical obstacles

CNOSSOS_P2P_SPECTRUM_WEIGHTING

enum CNOSSOS_P2P_SPECTRUM_WEIGHTING

Sound power frequency weighting.

Enumerator:

CNOSSOS_P2P_SPECTRUM_LIN

Spectrum is unweighted

CNOSSOS_P2P_SPECTRUM_dBA

Spectrum is A-weighted

Class Documentation

CNOSSOS_P2P_ENGINE

struct CNOSSOS_P2P_ENGINE

Hidden class structure containing the for path calculator.
Struct CNOSSOS_P2P_ENGINE is defined in file <CnossosPropagation.cpp>

Public Member Functions

CNOSSOS_P2P_ENGINE (const char *name)
bool **selectMethod** (const char *name)
bool **doCalculation** (void)

Public Attributes

ref_ptr< CalculationMethod > **method**
PropagationPath **path**
ElementarySource **source**
PropagationPathOptions **options**
PathResult **result**
bool **calculationDone**
bool **hasErrors**
std::string **errorMessage**

Constructor & Destructor Documentation

CNOSSOS_P2P_ENGINE::CNOSSOS_P2P_ENGINE (const char *_name_) [inline]

Constructor

Member Function Documentation

bool CNOSSOS_P2P_ENGINE::doCalculation (void) [inline]

Process the path.

bool CNOSSOS_P2P_ENGINE::selectMethod (const char *_name_) [inline]

Select the calculation method.

Member Data Documentation

bool CNOSSOS_P2P_ENGINE::calculationDone

Status of the calculation.

std::string CNOSSOS_P2P_ENGINE::errorMessage

Message associated with the last error encountered.

bool CNOSSOS_P2P_ENGINE::hasErrors

Status of the results

ref_ptr<CalculationMethod> CNOSSOS_P2P_ENGINE::method

Pointer to the calculation method.

PropagationPathOptions CNOSSOS_P2P_ENGINE::options

Configuration of the calculation engine.

PropagationPath CNOSSOS_P2P_ENGINE::path

Propagation path data

PathResult CNOSSOS_P2P_ENGINE::result

Output of the acoustical calculation.

ElementarySource CNOSSOS_P2P_ENGINE::source

Elementary source associated with the propagation path

Appendix D: Basic geometry operations

Digital representation of geometry relies on coordinates. Most often, coordinates are handled as ordered sets of numbers, written in matrix notation, using either single-row or single-column matrices to represent positions. Geometrical transformations, such as translations, rotations, projections,… are then formalized as operations on matrixes. This representation provides an elegant and compact mathematical formalism and may lead to highly efficient software implementations. An important drawback of matrixes is that they are nothing more than "structured sets of numbers", whose physical meaning (e.g. in terms of measured distances and angles) is far from obvious.

It is often more comprehensive to represent geometrical transformations by means of operations on vectors (including internal and external products), rather than relying solely on matrix algebra. High level programming languages, such as C+,+ make it easy to represent and expose the concepts of vector algebra directly, thereby hiding the underlying matrix operations to the user of the software.

The Cnossos-EU propagation software builds on a small library developed by CSTB. This library implements representation of vectors in two and three dimensions and operations on such vectors, including:

- sum and difference of vectors
- multiplication of vectors and real numbers
- dot product of two vectors
- external product of two vectors

Vector algebra can then be used for building higher level functionality such as conversions between different coordinate systems, projection of a point onto a plane or a line, calculation of the shortest distance between a point and a plane, convexity testing etc…

The basic operations on vectors are defined in the file "Geometry3D.h". All complex operations on propagation paths, as implemented in the file "PropagationPath.cpp", are expressed in terms of basic vector operations.

This appendix provides an overview of the basic operations on vectors and gives some indications on the way textual descriptions of geometrical operations as found in the reference documents are converted into equivalent operations on vectors and then into software code.

C.1 Positions, vectors and coordinates

Any Euclidean space E^n $(n = 1, 2, 3, \dots)$ becomes a vector space by defining an origin O and three orthogonal directions. Formally, a position P in E^n becomes a vector $\vec{p}$ by the mapping

$P \to \vec{p} : \vec{p} = (OP) = P - O$. Similarly, one can convert back vectors into positions by the inverse mapping $\vec{p} \to P : P = O + \vec{p}$.

For the purpose of the demonstration we limit ourselves to the case n=3 and denote the three orthogonal direction as X, Y and Z.

By constructing vectors of unit length along the selected orthogonal directions, any vector $\vec{p}$ can be uniquely identified with the set of numbers (x, y, z), such that:

$$\vec{p} = x.\overrightarrow{e_x} + y.\overrightarrow{e_y} + z.\overrightarrow{e_z}$$

It may be noted that, whereas P represents a unique position in the original Euclidean space (i.e. in the real world), its vector representation is not unique because it depends on the choice of both the origin and the orthogonal unit vectors. Formally, we can express this dependency as:

$$P = O + x.\overrightarrow{e_x} + y.\overrightarrow{e_y} + z.\overrightarrow{e_z}$$

C.2 Operations on vectors

All algebraic operations on vector can be expressed using coordinates. I.e. given two vectors

$$\overrightarrow{p_1} = x_1.\overrightarrow{e_x} + y_1.\overrightarrow{e_y} + z_1.\overrightarrow{e_z}$$

$$\overrightarrow{p_2} = x_2.\overrightarrow{e_x} + y_2.\overrightarrow{e_y} + z_2.\overrightarrow{e_z}$$

and some real numbers α_1 and α_2, one can construct new vectors by taking linear combinations of the original ones:

$$\vec{q} = \alpha_1.\overrightarrow{p_1} + \alpha_2.\overrightarrow{p_2} \quad = (\alpha_1 x_1 + \alpha_2 x_2).\overrightarrow{e_x} + (\alpha_1 y_1 + \alpha_2 y_2).\overrightarrow{e_y} + (\alpha_1 z_1 + \alpha_2 z_2).\overrightarrow{e_z}$$

Similarly, the dot and cross product of vectors can be expressed using coordinates:

$$\overrightarrow{p_1}.\overrightarrow{p_2} = x_1.x_2 + y_1.y_2 + z_1.z_2$$

$$\overrightarrow{p_1} \times \overrightarrow{p_2} = (y_1.z_2 - y_2.z_1).\overrightarrow{e_x} + (z_1.x_2 - z_2.x_1).\overrightarrow{e_y} + (x_1.y_2 - x_2.y_1).\overrightarrow{e_z}$$

In terms of geometrical quantities, we have:

- the length of a vector is given by:

$$\|\overrightarrow{p_1}\|^2 = \overrightarrow{p_1}.\overrightarrow{p_1} = {x_1}^2 + {y_1}^2 + {z_1}^2$$

- the distance between positions is equal to the length of the vector connecting these two positions:

$$dist(P_1, P_2) = \|\overrightarrow{p_1} - \overrightarrow{p_2}\|$$

- the angle between two vectors is given by:

$$\cos\theta = \frac{\overrightarrow{p_1}.\overrightarrow{p_1}}{\|\overrightarrow{p_1}\|\|\overrightarrow{p_2}\|} = \frac{x_1.x_2 + y_1.y_2 + z_1.z_2}{\sqrt{({x_1}^2 + {y_1}^2 + {z_1}^2)}\sqrt{({x_2}^2 + {y_2}^2 + {z_2}^2)}}$$

- two non-zero length vectors are perpendicular if and only if $\overrightarrow{p_1}.\overrightarrow{p_1} = 0$

- the vector $\overrightarrow{p_1} \times \overrightarrow{p_1}$ is perpendicular to both $\overrightarrow{p_1}$ and $\overrightarrow{p_1}$:

$$(\overrightarrow{p_1} \times \overrightarrow{p_2}).\overrightarrow{p_1} = (\overrightarrow{p_1} \times \overrightarrow{p_2}).\overrightarrow{p_2} = 0$$

- the vector $\overrightarrow{p_1} \times \overrightarrow{p_1}$ has a length equal to the area of the diamond shape formed by these vectors:

$$\|\overrightarrow{p_1} \times \overrightarrow{p_2}\| = \|\overrightarrow{p_1}\|.\|\overrightarrow{p_2}\|.\sin\theta$$

- two non-zero length vectors are aligned if and only if $\|\overrightarrow{p_1} \times \overrightarrow{p_2}\| = 0$.

Note: in our C++ implementation, the dot and cross products are implemented using the predefined operators "*" and "^". This allows writing such things as:

```
#using namespace Geometry3D ;

void f (Point3D p1, Point3D p2, Point3D p3)
{
    Vector3D q1 = p2 – p1 ;
    vector3D q2 = p3 – p1 ;
    double   dot_prod = q1 * q2 ;
    Vector3D cross_prod = q1 ^ q2 ;
    double   det = (p1 ^ p2) * p3 ;
    ...
}
```

C.3 Coordinate transformations

Implementing coordinate transformations by means of matrix operations often causes headaches and most people get thing right by means of "try and error". Using vector notation provides an elegant and comprehensive solution to this problem.

First, we note that if P is a position represented by a vector $\vec{p}$ in the Cartesian coordinate system OXYZ, then we recover its coordinates from:

$$\begin{cases} x \equiv \vec{p}.\overrightarrow{e_x} \\ y \equiv \vec{p}.\overrightarrow{e_y} \\ z \equiv \vec{p}.\overrightarrow{e_z} \end{cases}$$

Now if O'X'Y'Z' is another Cartesian coordinate system with origin O' and unit vectors $\overrightarrow{e'_x}, \overrightarrow{e'_y}, \overrightarrow{e'_z}$ then the position P can be expressed with respect to these as:

$$P = O + \vec{p} = O' + \overrightarrow{p'}$$

$$\overrightarrow{p'} = x'.\overrightarrow{e'_x} + y'.\overrightarrow{e'_y} + z'.\overrightarrow{e'_z}$$

Formal equality of both expressions for the unique position P leads to these simple expressions for the coordinates in the second coordinate system:

$$\overrightarrow{o'} = O' - O$$

$$\begin{cases} x' = \overrightarrow{(p - o')}.\overrightarrow{e'_x} \\ y' = \overrightarrow{(p - o')}.\overrightarrow{e'_y} \\ z' = \overrightarrow{(p - o')}.\overrightarrow{e'_z} \end{cases}$$

Coordinate transformations are intensively used in the Cnossos-EU propagation modules, and provide particularly useful for determining distances and heights with respect to ground planes and/or faces of diffracting wedges.

C.4 Lines and planes

Two non-identical points P_1, P_2 in a two or three-dimensional Euclidean space define a straight line. Each point on the line can be written uniquely as:

$$\vec{p} = \overrightarrow{p_1} + \alpha\ (\overrightarrow{p_2} - \overrightarrow{p_1})\ ;\ \alpha \in \mathbb{R}$$

The mapping $\alpha \in \mathbb{R} \rightarrow P \in E$ is a parametric representation for all points on the line. When restricting the parameter α to the interval $\alpha \in [0,1]$ we obtain a parametric representation for all points on the closed line segment $[P_1, P_2]$.

Similarly, three non-co-linear points P_1, P_2, P_3 in three-dimensional space define a planar surface S. Each point of S can be uniquely written as:

$$\vec{p} = \overrightarrow{p_1} + \alpha\ (\overrightarrow{p_2} - \overrightarrow{p_1}) + \beta(\overrightarrow{p_3} - \overrightarrow{p_1})\ ;\ \alpha, \beta \in \mathbb{R}$$

The mapping $(\alpha, \beta) \in \mathbb{R}^2 \rightarrow P \in E$ is a parametric representation for all points of S.

By considering the vector $\vec{n} = (\overrightarrow{p_2} - \overrightarrow{p_1}) \times (\overrightarrow{p_3} - \overrightarrow{p_1})$, which is, by definition, perpendicular to the plane S, we can now write down an explicit equation for all points of S:

$$P \in S \Leftrightarrow (\vec{p} - \overrightarrow{p_1}).\vec{n} = 0$$

Consider a plane $S: (\vec{x} - \overrightarrow{x_0}).\vec{n} = 0$ and the line $L: \vec{x} = \overrightarrow{p_1} + \alpha\ (\overrightarrow{p_2} - \overrightarrow{p_1})$. Then, the intersection of the line with the plane can be readily calculated by substituting the parametric representation of the line into the explicit equation of the surface:

$$((\overrightarrow{p_1} - \overrightarrow{x_0}) + \alpha\ (\overrightarrow{p_2} - \overrightarrow{p_1})).\vec{n} = 0$$

Under the condition that $(\overrightarrow{p_2} - \overrightarrow{p_1}).\vec{n} \neq 0$, i.e. that the line is not parallel to the plane, we have:

$$\alpha = -\frac{(\overrightarrow{p_1} - \overrightarrow{x_0}).\vec{n}}{(\overrightarrow{p_2} - \overrightarrow{p_1}).\vec{n}}$$

This is clearly easier to understand and to program than solving a 3x3 linear system by means of matrix inversion…

The orthogonal projection of a point $\vec{q}$ onto the plane S is readily obtained by considering the line $\vec{x} = \vec{q} + \beta.\vec{n}$, yielding:

$$\overrightarrow{q'} = \vec{q} - \frac{(\vec{q} - \overrightarrow{x_0}).\vec{n}}{\vec{n}.\vec{n}}.\vec{n}$$

Similarly, the mirror image of $\vec{q}$ through the place S is given by:

$$\overrightarrow{q''} = \vec{q} - 2\frac{(\vec{q}-\overrightarrow{x_0}).\vec{n}}{\vec{n}.\vec{n}}.\vec{n}$$

The above equations become even simpler if one first normalizes the vector $\vec{n}$ so that $\vec{n}.\vec{n} = 1$. In that case, $\beta = (\vec{q} - \overrightarrow{x_0}).\vec{n}$ is the (signed) distance from the point $\vec{q}$ to the (oriented) plane S.

C.5 Mean plane

The JRC-2012 method (as the original French NMPB-2008 method) calculates ground effects by replacing complex terrain profiles by a single equivalent mean ground plane.

The process for constructing the mean plane is fully documented in the reference document. On input, the algorithm takes a set of 2D coordinates $(x_i, y_i), i = 0,1, \dots N$ where x_i stands for the distance measured in the horizontal direction of the propagation plane and y_i the height of the terrain measured in the vertical direction. On output, the algorithm gives the parametric equation of the mean plane in the form $y = A.x + B$. The acoustical calculations then rely on distances measured either parallel or perpendicular to the mean plane. The evaluation of these distances is greatly simplified by means of vector formalism.

First, we transform the explicit representation of the mean plane into a parametric form:

$$\vec{x} = x.\overrightarrow{e_x} + y.\overrightarrow{e_y} = \vec{a}.t + \vec{b}\,; t \in \mathbb{R}$$

With:

$$\vec{a} = A.\overrightarrow{e_x} + \overrightarrow{e_y}$$

$$\vec{b} = B.\overrightarrow{e_x} + 0.\overrightarrow{e_y}$$

It is obvious that the point $\vec{b}$ is lying in the mean plane and the vector $\vec{a}$ is aligned with it.

From this, it is straightforward to construct a new orthonormal coordinate system aligned with the mean plane:

$$O' = \vec{b} = B.\overrightarrow{e_x} + 0.\overrightarrow{e_y}$$

$$\overrightarrow{e'_x} = \frac{\vec{a}}{\|\vec{a}\|} = \frac{A}{\sqrt{1+A^2}}.\overrightarrow{e_x} + \frac{1}{\sqrt{1+A^2}}\overrightarrow{e_y}$$

$$\overrightarrow{e'_y} = \frac{-1}{\sqrt{1+A^2}}.\overrightarrow{e_x} + \frac{A}{\sqrt{1+A^2}}\overrightarrow{e_y}$$

In order to obtain, for a given point P, the distance and the height relative to the mean plane, it is sufficient to calculate its coordinates in the new coordinate system, i.e.

$$d_P = (P - O').\overrightarrow{e'_x}$$

$$h_P = (P - O').\overrightarrow{e'_y}$$

Similarly, the image of P with respect to the mean plane is given by:

$$P' = P - 2.h.\overrightarrow{e'_y}$$

These operations are implemented in file "MeanPlane.cpp". Within the acoustical calculations, any mean plane is represented as an instance of the MeanPlane class.

A mean plane is constructed from an ordered sequence of 2D points. The class constructor first evaluates the parametric equation of the mean plane (see function getMeanPlaneCoefficients) and converts these into a vector representation with a local origin and local unit vectors aligned with the mean plane.

The MeanPlane class contains three additional methods:

- the "project" method converts coordinates in the propagation plane into local coordinates; i.e. after projecting, the new x value represents a distance measured parallel to the mean plane and the new y value the height measured perpendicular to the mean plane.
- the "unproject" method converts local coordinates relative to the mean plane back into global coordinates relative to the propagation plane.
- the "image" method takes the image of a point in the propagation plane with respect to the mean plane. Basically, this is implemented as a combination of the "project" and "unproject" methods, changing the sign of the y-component of the local coordinates in between.

The "project" and "image" methods have an additional Boolean parameter that when set to true will project the point onto the mean plane in case a negative height is found, i.e. "*if the equivalent height of a point becomes negative,… a null height is retained and the equivalent point is then identical with its possible image if there is diffraction*" (see VI.4.3.b, page 85).

C.6 Local coordinates associated with sources

Within the source models, directivity is expressed in a local coordinate system attached to the source. In the propagation modules, directivity must be evaluated for arbitrary propagation paths and arbitrary source locations and orientations. Trying to solve this problem by means of operations on local/global Euler angles is difficult because there is no simple algebra defined on angles. On the other hand, the problem can be solved very efficiently by means of vector calculations and explicit construction of the local (source related) coordinate-reference-system in terms of the global (project related) coordinates.

In general, positioning and orienting a local coordinate system offers 6 degrees of freedom, 3 for the position and 3 more for the orientation. Most often, some of this information is missing from the input data sets and a simplified construction should be envisaged. This simplified construction uses the position of the source as the new origin and a single directional vector for the possible rotations of the coordinate system. The choice of the directional vector depends on the type of source geometry.

For a point source, orientation is determined by a user-defined main direction of propagation (often defined in terms of Euler angles). It is assumed that this direction corresponds to the X direction of the local coordinate system attached to the source. The construction of the local coordinate system is defines as follows:

1) the user-defined direction $\vec{d}$ is taken as the new X-axis

$$\overrightarrow{e'_x} = \frac{\vec{d}}{\|\vec{d}\|}$$

2) the new Y axis is lying in the horizontal plane and perpendicular to $\overrightarrow{e'_x}$

$$\overrightarrow{e'_y} = \frac{\overrightarrow{e_z} \times \overrightarrow{e'_x}}{\|\overrightarrow{e_z} \times \overrightarrow{e'_x}\|}$$

3) the new Z axis is perpendicular to both $\overrightarrow{e'_x}$ and $\overrightarrow{e'_y}$

$$\overrightarrow{e'_z} = \overrightarrow{e'_x} \times \overrightarrow{e'_y}$$

It is obvious from the construction that $(\overrightarrow{e'_x}, \overrightarrow{e'_y}, \overrightarrow{e'_z})$ form a new orthonormal, right-handed, coordinate system and that the new $\overrightarrow{e'_z}$ vector is pointing upwards, i.e. that $\overrightarrow{e'_z}.\overrightarrow{e_z} \geq 0$.

For a line source, it is assumed that the Y axis is aligned with the trajectory of the source and that the X axis is both horizontal and perpendicular to the trajectory. This implies:

1) the tangent to the trajectory $\vec{t}$ is taken as the new Y-axis

$$\overrightarrow{e'_y} = \frac{\vec{t}}{\|\vec{t}\|}$$

2) the new X axis is lying in the horizontal plane and perpendicular to $\overrightarrow{e'_y}$

$$\overrightarrow{e'_x} = \frac{\overrightarrow{e'_y} \times \overrightarrow{e_z}}{\|\overrightarrow{e'_y} \times \overrightarrow{e_z}\|}$$

3) the new Z axis is perpendicular to both $\overrightarrow{e'_x}$ and $\overrightarrow{e'_y}$

$$\overrightarrow{e'_z} = \overrightarrow{e'_x} \times \overrightarrow{e'_y}$$

An area source is handled similarly to a point source with the user-defined main direction taken equal to the normal to the surface.

For all cases, the direction of propagation in the local coordinates is obtained from the global coordinates by means of the transformations defined in section C.3.

Note that, although they are not actually used in the test software[2] as provided under this project, these functionalities are implemented and available to programmers building full applications based on the libraries. In the latter case, the direction of propagation will be extracted from the propagation path and passed as a parameter to the source module which then returns apparent sound power, i.e. sound power per unit angle as radiated in the direction of propagation.

The software libraries contain all the necessary code for this:

- The construction of the source-related coordinate systems for different types of sources is implemented in the file "SourceGeometry.cpp".
- The evaluation of the direction of propagation and its conversion to the source-related coordinate system is implemented as part of the noise calculations; see function CalculationMethod::getSoundPower in file CalculationMethod.cpp.
- The direction of propagation relative to the local coordinate system attached to the source is passed as a parameter to the ElementarySource::getSoundPower virtual function.

Note that, for the moment being, the direction of propagation is not actually used in the evaluation of the sound power because the ElementarySource class acts as an abstract placeholder for more realistic source modelling. Fully operational noise mapping software would use some kind of adapter classes, derived from the abstract ElementarySource class, which in turn make calls to the appropriate source modules from within the getSoundPower method.

C.7 Euler angles

Although this is not exempted of ambiguity and incoherency, it is common practice to express directivity in terms of Euler angles.

A normal vector is defined in terms of Euler angles by:

$$\begin{cases} x = \cos\theta \cos\varphi \\ y = \sin\theta \cos\varphi \\ z = \sin\varphi \end{cases}$$

The inverse transformation is given by:

$$\begin{cases} \cos\theta = \frac{x}{1-z^2} \\ \sin\theta = \frac{y}{1-z^2} \\ \sin\varphi = z \end{cases} \quad \text{where:} \quad \theta \in [0, 2\pi[\ ; \ \varphi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

It must be noted that the inverse transformation becomes undefined for $z = 1$ or $z = -1$.

[2] In the test software, coupling of source and propagation module is available only through XML files. In fully operational noise mapping software, the coupling is direct in the sense that the application makes calls to both the source and propagation modules.

The Geometry.h package contains functions which construct a normal vector from Euler angles and vice-versa, construct Euler angles from any vector (see classes Vector3D and EulerAngles).

Note that, if directivity is expressed by means of separate functions of the Euler angles, i.e. as $D(\theta,\varphi) = D_h(\theta).D_v(\varphi)$, the directivity function will equally become undefined for $\varphi = \pm\pi/2$. This problem might be overcome by considering a different interpolation scheme based on Euler angles relative to the vertical (YZ) plane. Consider:

$$\begin{cases} y = \cos\theta' \cos\varphi' \\ z = \sin\theta' \cos\varphi' \\ x = \sin\varphi' \end{cases}$$

Or equivalently:

$$\begin{cases} \cos\theta' = \frac{y}{1-x^2} \\ \sin\theta' = \frac{z}{1-x^2} \\ \sin\varphi' = x \end{cases} \quad \text{where:} \quad \theta' \in [0, 2\pi[\ ; \ \varphi' \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

Then define the three-dimensional directivity as:

$$D(x,y,z) = D_h(\varphi').cos^2\,\theta' + D_v(\varphi').sin^2\,\theta'$$

This function is well defined for all values of (x,y,z) under the condition that $D_h(\varphi') = D_v(\varphi')$ for $\varphi' = 0$ and $\varphi' = \pm\pi/2$. Moreover, one has:

$$D(\theta, \varphi = 0) \equiv D_h(\varphi' = \theta)$$

$$D(\theta = 0, \varphi) \equiv D_v(\varphi' = \varphi)$$

I.e. the interpolated 3D directivity coincides with the 2D directivity functions defined in the horizontal and vertical plane (as one would expect).

Appendix D: Geometrical analysis of propagation paths

Path finder algorithms available in commercial software are able to construct propagation paths including reflections from vertical walls (barriers or buildings) and/or diffractions around vertical edges.

The acoustical calculations take on input a description of the propagation path in 3D (see figure below). The path consists of positions in the horizontal plane and associated height information. These locations correspond to the intersection of the propagation plane with the "boundary" of the 2.5D geometrical model, i.e. the interface between the fluid (air) in which sound propagates and the solid part beneath it. The solid part is made of ground, buildings, walls, barriers etc…

The path finder also marks the characteristic points of the path, including:

- The footprint of the source projected on the boundary,
- The footprint of the receiver projected on the boundary,
- The footprint of vertical obstacles that act as specular reflectors,
- The footprint of the vertical edges that act as laterally diffracting edges.

For each characteristic point, additional information is required to fully describe its location and extent. This information is encoded as height relative to the underlying footprint position.

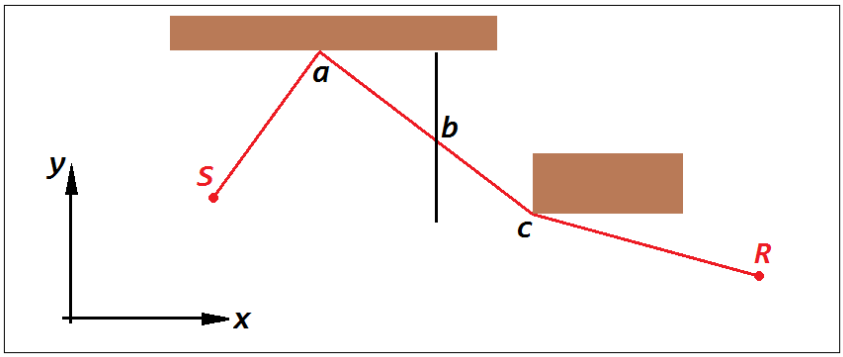


Figure a: propagation path in the horizontal plane

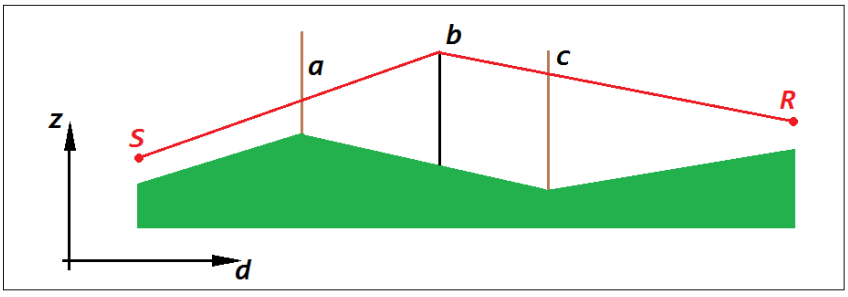


Figure b: unfolded propagation path in the vertical plane

By definition, the projection of the propagation plane on the horizontal consists of a set of straight line segments connected one to the next at the characteristic points.

The path calculator calculates additional attenuations for reflections from vertical obstacles (e.g. point *a*) and diffractions by vertical edges (e.g. point *c*). It then "unfolds" the path into a 2D problem in the vertical plane (see figure) and passes the simplified problem over to the selected point-to-point module.

However, not all propagation methods will be able to handle propagation paths of arbitrary complexity that can be constructed by modern path finder algorithms. It may be noted that the path shown above is **not** a valid path for any of the three propagation methods. According to the JRC-2012 reference report because it contains diffractions by both horizontal and vertical edges (see sections VI.3.2 and. VI.4.4). For similar reasons, the path is not acceptable as a valid input to the ISO 9613-2 method. Neither can it be evaluated by means of the JRC 2010 method because that method does not include lateral diffraction at all.

It is therefore essential to check the validity of propagation paths before sending them on to the acoustical calculations. Because this checking relies on common functionality found in all propagation methods, it is implemented in a single module, i.e. in file *PropagationPath.cpp*.

Note that, often commercial software include path finding algorithms that can guarantee – by construction – that propagation path output are conforming to the input specifications for the acoustical calculation methods. In this case, the validity checks are not really needed and can be disabled, thus reducing the computation times.

Because the Cnossos-EU software is intended to be used with input data sets generated by hand, all validity checks are enable by default.

For a complete list of checks that can be enabled and/or disables, see section 3.12 of this document. Details on the implementation of the most important checks are given in the next sections. For more details, check the source code starting at file *PropagationPath.cpp* line 60.

```
bool PropagationPath::analyze_path (PropagationPathOptions const& options)
{
	if (! remove_barriers()) return false ;
	if (! check_source_receiver (options)) return false ;
	if (! setup_horizontal_path (options)) return false ;
	if (! setup_vertical_path (options)) return false ;
	return true ;
}
```

D.2 Straight line propagation

In between two critical points, the path must satisfy the "straight line" condition.

Letting the projection of the path on the horizontal plane consists of points $P_i(x_i, y_i), i = 0 \dots N$ and if $P_{k1}, P_{k2}, 0 \leq k_1 < k_2 \leq N$ are two successive critical points, the library creates a local coordinate system aligned with the segment $[P_{k1}, P_{k2}]$:

$P_{k1}(x' = 0, y' = 0)$

$P_{k2}(x' = 0, y' = d)$

and checks, for each intermediate point $P_j, k_1 < j < k_2$, $P_j(x' = u_i, y' = v_i)$ the relative distance to the straight line segments, allowing for a small alignment error:

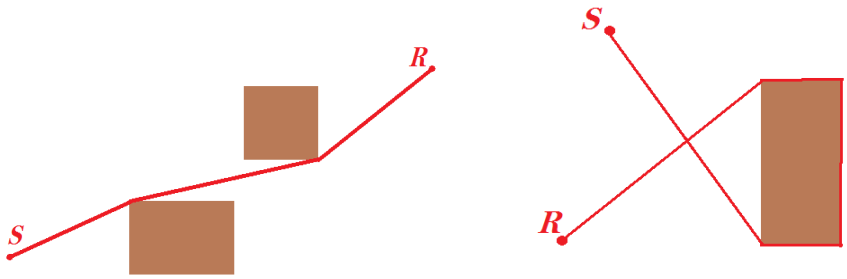
$$|v_i| < 0.005\sqrt{u_i(d - u_i)}$$

D.3 Lateral diffraction

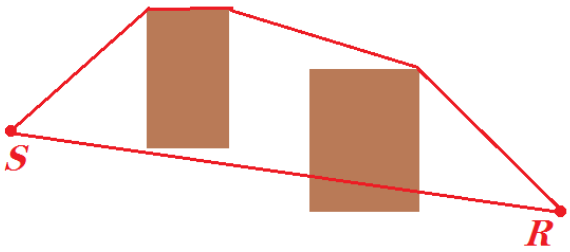
It is assumed that laterally diffracted paths correspond to paths deflected into the shadow zone formed by the source and the vertical edge of the diffracting object. As the calculation module has no access to the full 3D geometry of the diffracting wedge, this assumption cannot be validated within the software.

In case of multiple lateral diffractions, both the ISO 9613-2 and JRC-2012 methods assume that these can be calculated similarly to the methods laid down for diffraction over multiple obstacles, i.e. assuming that the geometrical path difference is representative for the diffraction effect.

Although modern path finder algorithms may be able to construct propagation paths as shown below, the ISO and JRC-2012 methods are not designed to calculate such complex situations. Blind application of the formulas may lead to physically unrealistic results that were most likely not intended by the authors of the methods.



In order to apply the diffraction formulas for the vertical plane to a problem in the horizontal plane, one must be able to imagine that there exists some kind of base line, equivalent to the ground plane in the vertical plane. A good candidate for this base line is naturally the direct line connecting the source and the receiver, as illustrated below:



The criterion for a valid laterally diffracted path may then be formulated as following: *the projection of the laterally diffracted path on the horizontal plane, closed by the direct line connecting the source and the receiver form a convex polygon*.

The library simply constructs the 2D loop containing the source, the receiver and the critical points marked as laterally diffracting edges and then checks for convexity (see function *is_convex* in file Geometry3D.h).

Note that this algorithm does not work in case the path contains both laterally diffracting edges and reflections from vertical wall. These situations anyhow fall outside the scope of all three propagation methods and are eliminated anyhow (see section D.7).

D.4 Constructing the convex hull

After validating the path in the horizontal plane, the path is unfolded into a 2D problem in a single vertical plane:

$$P_i(x_i, y_i, z_i) \rightarrow P_i(d_i, z_i) \;;\; i = 0 \ldots N$$

$$d_0 = 0$$

$$d_i = d_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \;\;;\; i = 1 \ldots N$$

The construction of convex hull is based on the recursive search for the most diffracting edge where it is assumed that the importance of diffraction can be solely determined from the geometrical path length difference (see e.g. ref [7] to appendix E).

The algorithm starts by setting $m = 0, n = N$ and by adding the source and receiver heights to the appropriate locations in the path; e.g. replace $P_0(d_0, z_0)$ by $P_0(d_0, z_0 + h_S)$ and $P_N(d_N, z_N)$ by $P_N(d_N, z_N + h_R)$.

Then, the following process is applied recursively:

For each $i = m + 1 \ldots n - 1$, calculate

$$\delta_i = \pm(|P_m P_i| + |P_i P_m| - |P_m P_n|)$$

With

$$|P_i P_j| = \sqrt{(d_i - d_j)^2 + (z_i - z_j)^2}$$

And the negative sign is taken if the point $P_i$is located below the line $P_m P_n$.

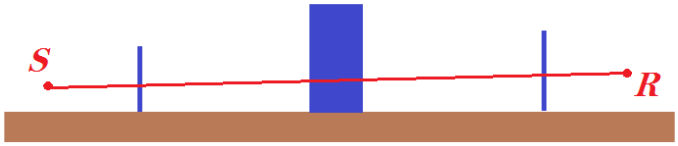
Locate the most diffracting edge $P_k : \; \delta_k = \max\{\delta_i \;;\; i = m + 1 \ldots n - 1\}$

If $\delta_k > 0$ mark the position as a diffracting edge and repeat the process on the left and the right of P_k; i.e. for $m' = m, n' = k$ and $m' = k, n' = n$.

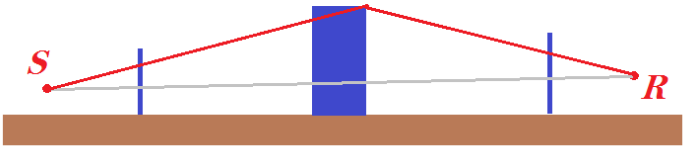
The process is repeated until all $\delta_k \leq 0$.

If the recursive search is terminated with $\delta_k \leq 0$ and $m = 0, n = N$, the edge P_k is marked as a potential diffracting edge below the line of sight.

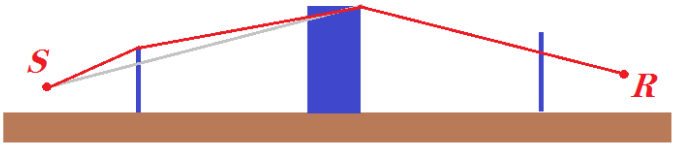
Step 0:



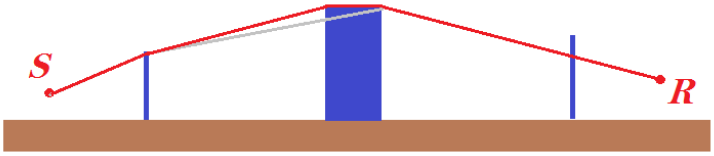
Step 1:



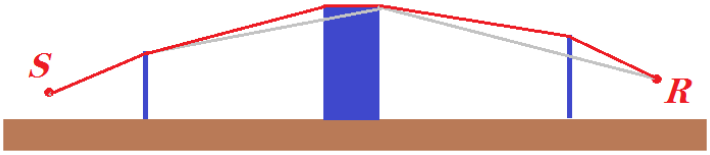
Step 2:



Step 3:



Step 4:



Note that at most one diffracting edge below the line of sight will be selected. This edge may be used in the acoustical calculations to generate a diffraction effect for a receiver in direct view of the source (i.e. in the illuminated region).

D.5 Upper and lower bounds of vertical extensions

While constructing the convex hull, vertical extensions representing reflecting obstacles or laterally diffracted edges are ignored, i.e. considered as perfectly transparent.

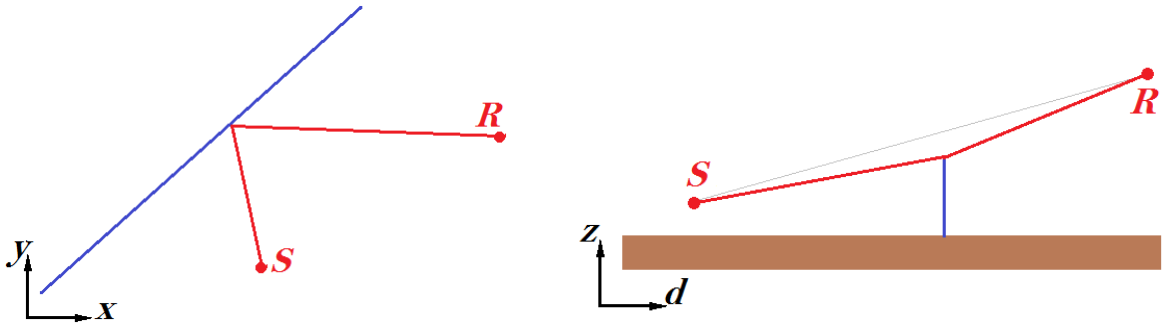
It may be noted that the convex hull is a faithful representation of the Fermat path connecting the source to the receiver (in the original 3D model i.e. before unfolding the propagation path). The convex hull, as a ray path, respects the rules of specular reflections and Keller diffraction; i.e. the angle of the incident ray equals the angle of reflected/diffracted rays.

Therefore, the intersection of convex hull with the fictive segments representing the vertical extensions faithfully represents the specular reflection point and/or the top of the Keller cone for diffracted rays. The height of the critical point can therefore be determined as:

$$z_k = z_m + (z_n - z_m)\,\frac{d_k - d_m}{d_n - d_m}$$

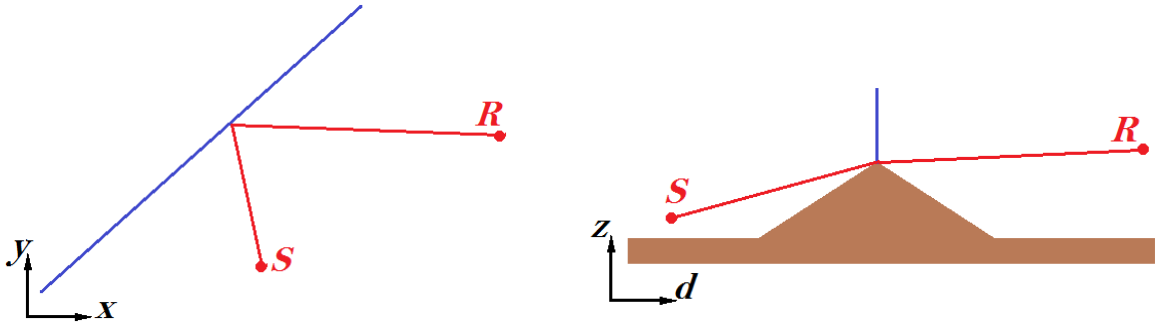
The ISO 9613-2 standard requires that the specular refection point must fall inside the limits of the reflecting surface, or, in other words, that the reflected ray paths obey the laws of optical acoustics.

The NMPB and Harmonoise models do not include such a restriction. As the specular reflection point approaches (and exceeds, see figure below) the upper limit of the reflecting obstacle this is compensated by means of retro-diffraction or Fresnel weighting, thus providing a continuous solutions for receivers inside/outside the range of optically reflected ray paths.



Similarly, if the lower edge of the obstacle is part of the convex hull, the ray path will be defected at that location, indicating that the equal-angle criterion for specular reflection is not respected (see figure below).

In this case, the ISO 9613-2 method will reject the path as being non conformant to the rules of optical acoustics whereas the NMPB and Harmonoise may accept the path. In the latter case, the lower edge of the obstacle will be considered as a diffracting edge in the unfolded propagation path and extra attenuation will be automatically taken into account.



Although none of the standards specify that Keller's construction of ray paths should be respected for laterally diffracting edges, we assume that this case can be handled similarly to the case of reflected paths.

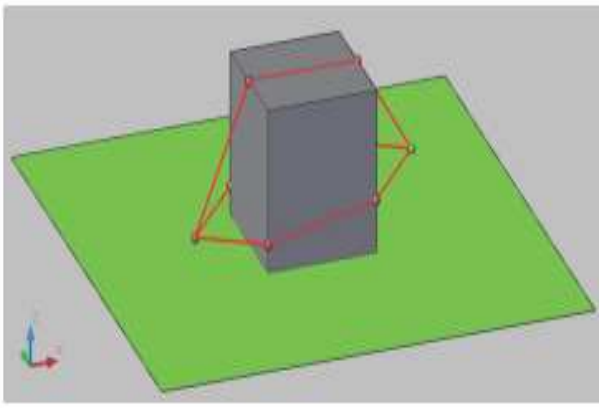
The application of the height criteria is controlled by the *checkHeightLowerBound* and *checkHeightUpperBound* options as explained in section 3.12.). Both options are forced to true in case of ISO 9613-2, otherwhis they are set to false by default but can be enabled/disabled by application software (e.g. for increasing comparability). The tests apply to both reflections and lateral diffractions.

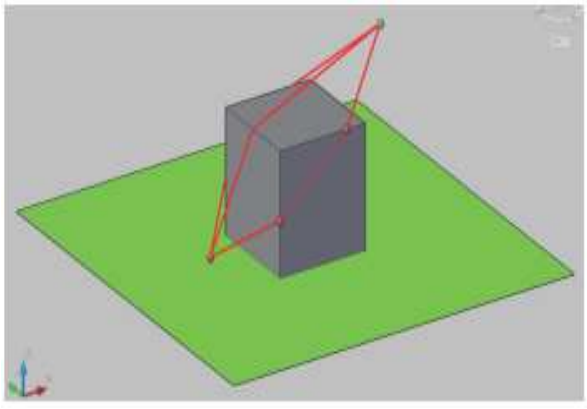
D.6 Complex path geometries

Complex paths are paths that may contain any number of diffraction and reflections. None of the three can handle arbitrary complex paths. Limitations of the different methods are indicated in the table below.

Degree of complexity	ISO 9613-2	JRC-2012	JRC-draft-2010
no lateral diffraction, any number of reflections and diffractions over top edges	yes	yes	yes
lateral diffraction, no reflections, no diffractions over top edges	yes	yes	no
lateral diffraction + at least one reflection	no	no	no
lateral diffraction + at least one diffraction over top edge	no	no	no

Note: the extended interpretation of diffracted paths as proposed in ISO/DTR 17534-3 is not considered for the moment being. Paths diffracted solely by horizontal or vertical edges (left picture below) are valid input for the point-to-point calculations. Complex paths diffracted by both horizontal and vertical edges (right picture) are not. It may be noted that the construction of such complex path requires 3D ray-tracing techniques whereas most noise mapping software solely relies on 2.5D path construction[3].





[3] From a purely physical point of view, application of simple diffraction formulas (based solely on the path length difference) to such complex paths is questionable in terms of achievable accuracy. From a pragmatic point of view, it is questionable whether such paths are relevant for the purpose of large scale noise mapping.

Appendix E: Implementation details

E.1 Meteorological conditions

During phase A of the CNOSSOS-EU project, intensive discussions have taken place concerning the pro and cons of the selected candidate propagation methods. Much less effort has been spent on the integration of meteorological aspects into the common framework for harmonized noise mapping methods. It has been more or less implied that the meteorological approach would follow from the choice of the propagation method.

The subject is often misunderstood due to a lack of clear definitions. The Harmonoise/Image projects proposed the following harmonised definitions[4]:

- The term "propagation conditions" is used solely to describe the acoustical effects of specific meteorological conditions as those may exist for a relatively short period of time, e.g. when measurements are being carried out. From a purely physical point of view, propagation conditions are fully described by the vertical profile of the effective sound speed.
- The term "meteorological conditions" refers to specific weather conditions, i.e. wind direction, wind speed, cloud cover,… that directly influence the propagation conditions. In physical terms, meteorological conditions are fully described by the vertical profiles of wind speed, wind direction, temperature,…
- The term "meteorological weighting" refers to the variability of propagation conditions over time and the effect this has on long-time averaged noise indicators such as LDEN or Lnight. In general meteorological weighting is related to climatologically relevant statistics of weather conditions, possibly averaged over many years.

The propagation methods allow for the estimation of the excess attenuation along a propagation path under specific propagation conditions.

- The ISO propagation model allows for calculations under a single propagation condition, named "moderate downwind conditions" in the text, but generally assimilated to any sound speed profile that result in moderate favourable propagation effects.
- The NMPB propagation model allows for calculations under two specific propagation conditions, namely homogenous and favourable. The reference text from Sétra allows linking these propagation conditions to specific meteorological conditions in terms of wind speed, wind direction and thermal stability classes.

[4] In order to avoid any confusion between "meteorological conditions" and "propagation conditions", note that a single meteorological condition can lead to two different (or even opposite) propagation conditions. For instance. if sound propagates from A to B under downwind (favourable) conditions, then, at the same time, propagation from B to A will be unfavourable. Similarly, very different meteorological conditions can lead to almost equivalent propagation conditions. This is the reason why efficient evaluation of long term averaged noise levels proceeds by first projecting the meteorological (weather) conditions on a limited set of propagation conditions and then evaluates the propagation effects under these conditions.

- The Harmonoise propagation model is able to predict propagation of sound under any meteorological condition that can reasonably be assimilated with a linear gradient of the effective sound speed. The Harmonoise and Imagine projects have proposed several ways for linking meteorological observations to sound speed profiles and for the transformation of arbitrary sound speed profiles into equivalent linear gradients; the choice of the most appropriate and efficient way being influenced by the very nature of the available meteorological data (e.g. ranging from simple surface observations, up to meteorological towers, balloon soundings, climatological data sets, advanced prediction methods,…).

In phase A of the CNOSSOS-EU project, the experts from WG5 investigated the equivalence between the three methods by stating that homogeneous conditions correspond to the absence of any sound speed gradient and that favourable propagation conditions (in either ISO or NMPB) could be assimilated with a linear sound speed gradient of 0.07 s^{-1}; a value which is in line with the proposals of the French expert group working on behalf of Sétra.

The findings of the WG are summarised in the table below:

Propagation condition	Favourable	Homogeneous	Other
Gradient	A =0 s^{-1}	A =0.07 s^{-1}	Any linear gradient
ISO 9613-2	yes	no	no
NMPB-2008	yes	yes	no
Harmonoise	yes	yes	yes

In principle, each propagation method proposes an associated weighting model for the estimation of long term averaged noise indicators.

- The ISO method introduces a C_{meteo} correction factor which relates long-term-averaged noise levels to downwind conditions. Although the standard specifies that this correction term might be related to local weather statistics, it is generally considered a user-defined input value to be filled in by experts based on experience and/or good practice.

$$L_{p,LT} = L_{p,F} - C_{meteo}$$

- The NMPB method has been the first (back in 1992) to introduce an objective characterisation of long term averaged noise levels based on the frequency of occurrence of favorable propagation conditions in a given direction of propagation. The values of the model were derived from long-term recordings of past weather conditions.

$$L_{p,LT} = 10.\log\left(p_F.10^{Lp,F/10} + (1-p_F).10^{Lp,H/10}\right)$$

- The Harmonoise model was first validated using by means of 120 different weather conditions (i.e. 3 wind speeds, 8 wind directions and 5 thermal stability classes) where the frequency of occurrence of each of these conditions was directly obtained from long-term weather records. This method was used for validation purposes but deemed impracticable for the purpose of large scale noise mapping.

- The Imagine project found that long-term averaged noise levels could be accurately determined using at most 5 propagation classes, each propagation class being defined in terms of an equivalent linear gradient. As in the NMPB approach, the weighting factors of the model actually represent the frequency of occurrence of each specific propagation condition and can easily be linked to climatological recordings.

$$L_{p,LT} = 10.log \sum_{i=1}^{N} p(A_i).10^{Lp,i/10}$$

It is obvious that the NMPB model is a subset of the IMAGINE model insofar that it considers only two propagation classes[5].

On the other hand, the ISO weighting model is compatible with any of the three propagation models insofar that they all allow for the evaluation of noise levels under "moderate favourable propagation conditions".

This leads to the following compatibility matrix:

Meteorological weighting	ISO-9613-2	NMPB	IMAGINE
Number of classes	$N = 1$	$N = 2$	$1 \leq N \leq 5$
Parameters	C_{meteo}	p_{fav}	$p(A_i)$
ISO 9613-2	yes	no[6]	no
NMPB-2008	yes	yes	no[7]
Harmonoise	yes	yes	yes

It may be noted that the collection of long time weather statistics was never considered part of the Harmonoise and Imagine projects; neither did Phase A of the Cnossos-EU project investigate this topic in detail.

Because the reference JRC report (version 2010) does not explicitly make reference to the IMAGINE meteorological weighting model and no specific provisions were made in the service contract, it has been decided not to implement it as part of the propagation library.

Pragmatically, this means that the implementation of the Harmonoise propagation model in CNOSSOS-EU is restricted[8] by the compatibility with the NMPB meteorological weighting model,

[5] The feasibility of a model using at most 5 propagation classes is fully in line with the findings of the French expert group who first proposed the U_iT_j grid to convert the 5x5 weather classes into 5 propagation classes (noted --, - , N, + and ++ in the original documents). For calculation purposes, it was then decided to combine classes in two groups: homogeneous (--,- and N) and favourable (+ and ++).

[6] The ISO propagation model can be used with the NMPB meteorological weightings assuming that the noise level under homogeneous conditions is negligible compared to favourable conditions. This is the assumption implemented in the current version of the library. Alternatively, one might use heuristic corrections in order to evaluate the effects of favourable versus homogeneous propagation conditions, as suggested in section 7.2 of this document.

[7] The NMPB method can be used with the general weighting model assuming that the noise levels in each class can be evaluated as either "favourable" or "homogeneous".

i.e. by retaining only two propagation classes with fixed values for the corresponding representative sound speed gradients ($A = 0s^{-1}$ or $A = 0.07\ s^{-1}$).

It is estimated that this approach may be sufficiently accurate for the purpose of strategic noise mapping and therefore not a determinant criterion for the selection of the harmonised methods. Moreover, the proposed approach is compatible with the recommended interim method and the guidelines of the Good Practice Guide, which suggest, in absence of local meteorological statistics, o use the following default values[9]:

Day time	$p_{fav} = 50\%$
Night time	$p_{fav} = 100\%$
Evening	$p_{fav} = 75\%$

In view of these considerations, the Cnossos-EU library was built so that all three propagation models can be combined with the two candidate meteorological models. As the expert working group did not provide any advice on the equivalence of the frequency of occurrence of favourable propagation conditions and the value of C_0, both values were kept as user-defined input values.

The proposed approach reflects the current state-of-the-art and best practices for the purpose of strategic noise mapping, while leaving definitive answers to later stage of the Cnossos-EU project, i.e. with respect to open issues not (yet) decided by the expert working groups:

- In order to increase comparability and consistency, an objective procedure should be established to link the frequency of occurrence of favourable propagation conditions to national (or preferably European) databases of weather conditions.
- If the ISO propagation method were to be selected as the future harmonised method for noise mapping, a harmonised procedure should be made available in order to link the C_0 correction term to weather statistics (directly) or to the frequency of occurrence of favourable propagation conditions (indirectly) in a coherent and comprehensive way all through Europe (unless the choice is made to leave the value to be determined by local or national authorities)

E.2 Sound power adaptation

There is an important difference between Harmonoise on one hand and the ISO 9613-2 or NMPB models on the other hand: the first takes free field sound power as its reference for calculating noise levels whereas the latter methods use "in situ" sound power on input.

The fact that "sound power" can have different meanings in different calculations methods has been put forward in IMAGINE WP7 and integrated in the database for industrial noise sources,

[8] As the full functionality is included in the distribution, software developers may use the advanced meteorological and climatological models from the Harmonoise/Imagine projects for purposes others than strategic noise mapping, e.g. in long-range propagation under specific weather conditions defined in terms of vertical gradients of wind speed, direction and temperature.

[9] These values were derived for use with NMPB version 1996. Because the definition of favourable propagation conditions has changed in NMPB version 2008, it might be necessary to revise these default values.

i.e. the database contains on indicator for the measurement conditions of the reported sound power values:

indicator	meaning
Free field	Sound power was measured avoiding any influence of reflecting surfaces, e.g. in an anechoic room, or sufficiently high above absorbing ground. For an omnidirectional source: $L_p = L_w - 10.\log\ (4\pi d^2)$
Hemispheric	Sound power was measured with the source placed on or near a hard reflecting surface. The determination of sound power assumes hemispherical spreading of sound energy: $L_p = L_I = L_w - 10.logS$ with $S = 2\pi d^2$, the area of the upper hemisphere surrounding the source.
Undefined	Measurement conditions unknown or not specified.

It may be noted that the ISO 9613-2 standard also mentioned this ambiguity and provides a D_Ω correction term that "accounts or sound propagation into solid angles less than 4π steradians". This means that the symbol L_W in the standard should actually be understood as "sound power radiated under free field conditions". This is consistent with the fact that when a point source is placed very near a hard surface, the sound level is increased by 6 dB which in the ISO approach is evaluated as an increase of sound power ($D_\Omega = +3\ dB$) and an increase due to the ground effect ($A_{ground} = -3\ dB$). The Harmonoise model on the other hand, does not account for the increase of sound power but instead predicts a $+6\ dB$ level increase due to the (coherent) reflection on the ground.

ISO standards (series 3740) on the other hand, recommend to measure and report sound power levels under operating conditions similar to those of the real sources. I.e. if a source is generally placed near a hard reflecting surface, the reported sound power level will include the correction term and:

$$L_{W,measurement} = L_{W,free\_field} + D_\Omega$$

When now $L_{W,measurement}$ is used on input to the ISO propagation model, there is no need to apply the correction for hemispherical radiation conditions a second time. As a consequence, it has become common practice to simply ignore the D_Ω correction and to use the "ISO 3740" compatible sound power levels on input of the modelling equations. This is specifically true for road vehicle sources which are systematically determined using the assumption of hemispherical radiation: the hard road surface beneath the vehicle being an integral part of the source, trying to revert to free field radiation does not make much sense.

As a consequence: both the ISO and the Harmonoise methods provide a coherent approach to the coupling of their respective source and propagation models. However, problems arise when it comes to comparing and/or combining source and propagation models from different origins.

Although it might be envisaged to implement the D_Ω factor as a (frequency dependent) input value to be defined by the end-user, it seems safer and easier to consider it as part of the propagation model. A paper published in 1958 by Ingard and Lamb [1] gives the theoretical relationship between free-field and hemispherical sound power. For an omnidirectional source at height z_S above hard ground:

$$\Delta L_w = L_{w,hemi} - L_{w,free} = 10.\log\left(1 + \frac{\sin(2kz_s)}{2kz_s}\right)$$

A slightly modified version of this formula is implemented as part of the path calculator, i.e. taken into account the nature of the ground beneath the source:

$$\Delta L_w = L_{w,hemi} - L_{w,free} = 10.\log\left(1 + \frac{\sin(2kz_s)}{2kz_s}\right).(1 - G_S)$$

The correction will be triggered if the definition of sound power does not match the assumptions of the propagation model, according to the following rules:

	Point-to-point propagation model	
Sound power indicator	ISO 9613-2 or NMPB	Harmonoise
Free field	$+\Delta L_w$	0 dB
Hemispherical	0 dB	$-\Delta L_w$
Undefined	0 dB	0 dB

Note that the correction is zero in case of soft ground ($G_S = 1$) and negligible small for a source sufficiently high above the ground ($k.z_S > 1$) ; e.g. if $z_S > 0.50m$, $\Delta L_w \cong 0\ dB(A)$.

E.3 Material properties

Sound propagation is influenced by the acoustical properties of the boundary between air and solid bodies (i.e. ground, buildings, barriers,…). Although it is possible to let the end-user enter such properties for each geometrical element in the model, it seems more appropriate to use a two-step method based on the selection of materials and the assignment of physical properties to these materials.

Different propagation models rely on different characterisations of the material's properties. Although it is possible to let the end-user define each characteristic independently one from another, it is preferable to have a coherent set of values for all properties and a set of basic rules that allow interfering missing values from known user input.

The implementation of material properties in the Cnossos-EU propagation software is a direct result of these design rules. Materials are stored in a shared database in central memory. Each material has a unique identifier (a name, stored as a short text) and a set of associated physical properties.

At system start up, the database is filled with a set of generic materials with predefined properties. This includes both materials applicable to ground (and roofs) and to vertical surfaces in the model (i.e. walls, barriers and façades).

The Cnossos-EU library uses the following material properties:

- The absorption factor G as defined in ISO-9613-2,
- The equivalent flow resistivity σ, expressed in kPa.s/m²,
- The frequency dependent complex impedance $Z(f)$,
- The frequency dependent absorption coefficient $\alpha_S(f)$ as defined in ISO 354 [2].

The minimal requirement for each material is to have an absorption factor G defined. All other properties are optional and can be either defined in terms of user-defined values or, if missing, evaluated in terms of the other properties.

For use with the Harmonoise model, a material should be assigned an impedance spectrum. Although the Harmonoise models allows for user-defined impedance values (obtained either by measurement or by one of the many models available in literature), it has been found that, for most applications, the Delany-Bazley one-parameter impedance model [4] provides a sufficient approximation for the impedance of ground.

$$Z(f) = 1 + 9.08\left[\frac{f}{\sigma}\right]^{-0.75} + 11.5\,j\left[\frac{f}{\sigma}\right]^{-0.73}$$

The Delany-Bazley oen-parameter impedance model is considered an integral part of the Harmonoise propagation model and implemented as such in the library. Therefore, the flow resistivity, although not directly used in the acoustical calculations, is considered an essential material property and stored as such in the database.

If the flow resistivity is not defined by the end-user, it is evaluated as a function of the absorption factor according to the table from the French NMPB standard. A simple formula that accurately approximates the values of the table is e.g.

$$\sigma = 20000.10^{-2G^{3/5}}$$

All three propagation models use the diffuse field absorption coefficient α_S when evaluating the effects of reflections from vertical obstacles. In principle, the absorption coefficient is a user-defined input value. If the value is not defined by the end-user, it will be determined analytically from the (complex) acoustic impedance by means of [5]:

$$\beta = \frac{1}{Z} = X + j.Y$$

$$\alpha_S = 8.X.\left[1 + \frac{X^2 - Y^2}{Y}tan^{-1}\frac{Y}{X^2 + X + Y^2} - X.ln\frac{(X+1)^2 + Y^2}{X^2 + Y^2}\right]$$

Of course, such converted values should be considered as first-order approximations and it is highly recommended in real-life situations to use measured acoustical properties instead. Note that all predefined materials have both sigma and G values defined. This is particularly true for the absorption values associated with noise barriers. Measurement of absorption values of noise

barrier in the lab is common practice and reported values in octave bands can be used in noise calculations.

All predefined materials are defined in terms of both G and Z values, as indicated in section 3.2 of this document. The classification for ground types was first proposed by the Nord-2000 project and later integrated as part of the Harmonoise and NMPB-2008 methods.

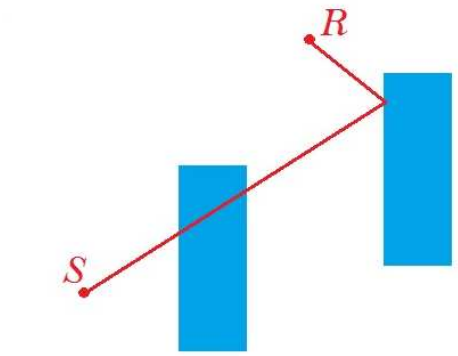
The classification for vertical wall coverings was proposed by the Imagine project. The default materials match the classification scheme given in EN 1793-1. For each class, the equivalent value of the flow resistivity was determined as the highest value that allowed for an absorption coefficient matching the minimal requirements of each class, in terms of the evaluation index DL_α. The conversions from flow resistivity into absorption coefficients were carried out as described in this section.

E.4 Fresnel weighting

The Harmonoise propagation model relies on Fresnel weighting in order to account for the finite height of vertical reflecting obstacles in the propagation path.

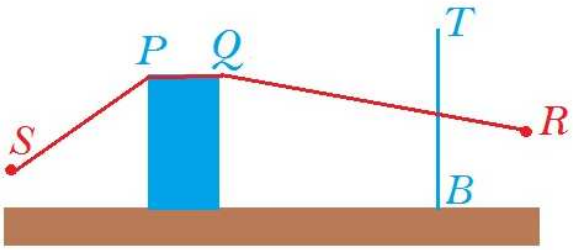
Although the deliverables from the Harmonoise and the Imagine project lay down the principles of Fresnel weighting, they do not provide a complete description of its application to complex propagation paths.

In order to illustrate the algorithmic implementation of Fresnel weighting in the Cnossos-EU software, consider the following example (see figure below). In general, a 2.5D path finder algorithm will construct reflected paths in the horizontal plane (see figure below) and then add height information to all relevant elements in the path.



At the reflection point, height information includes the altitude of both the foot point (B) and the he top (T) of the reflecting obstacle. In the unfolded propagation path, the reflecting position becomes the vertical line segment BT (see figure below).

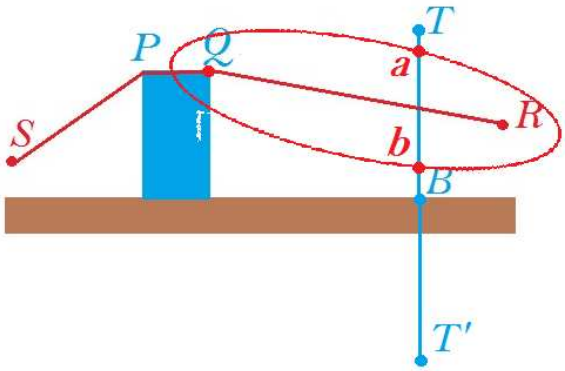
After constructing the Fermat path and, if required, eliminating non-specular reflections (see section D.3), the method determines the attenuation due to this reflection point and then removes it from the path.



For each reflection, the correction is calculated as:

$$A_{ref,i} = 10.\log(1-\alpha_i) + 10.logF_i$$

Where F_i is the fraction of the barrier lying inside the Fresnel zone associated with this reflection point and α_i the absorption coefficient of the material covering the reflecting surface.

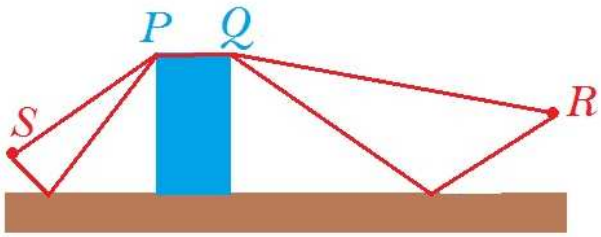


The Fresnel fraction is evaluated as follows:

- Let S' be the equivalent source to the left of the reflecting obstacle. If there is no diffraction in between the real source and the reflecting obstacle, then S' = S, otherwise, S' is the diffracting edge in between the source and the reflecting obstacle nearest to the latter. E.g. in the picture above, S' = Q
- Let R' be the equivalent receiver to the right of the reflecting obstacle. E.g. in the picture above, R' = R
- Construct the Frensel ellipse for the couple S',R' and a path difference $\delta = \frac{\lambda}{4}$
- Consider the image T' of the top of the reflecting obstacle with respect to the ground at the feet of the reflecting obstacle; i.e. in the picture |T'B'| = |BT|
- The Frensel ellipse cuts the straight line segment [T,T'] at two points; i.e. points a and b in the picture.
- The Fresnel fraction is defined as the size of the intersection of the finite segment TT' with the Fresnel ellipse compared to the size of the intersection of the Fresnel ellipse with the infinite straight line supporting the segment TT'.

I.e. in the picture $F_i = 1$ because the intersection of (a,b) with (T,T') is equal to (a,b)

After removing all reflecting obstacle from the path, the excess attenuation is calculated over the unfolded path, taking into account diffractions by horizontal edged and reflections on the ground. E.g. the model will account for diffractions at points P and Q (in that order) and for ground reflections on the segments (S,P), (P,Q) and (Q,R) – including the vertical segments at P and Q.



Note that the Fresnel zone is compared to the size of the reflecting obstacle including its reflection on the ground. This is required because the right-angled reflector at the point B does not produce any attenuation due to diffraction, i.e. it acts as a perfect mirror no matter the exact position of the specular reflecting point. In terms of ray acoustics, there is a single, continuous field produced by ray paths first hitting the ground, then the vertical reflector and ray paths first hitting the vertical reflecting, then the ground.

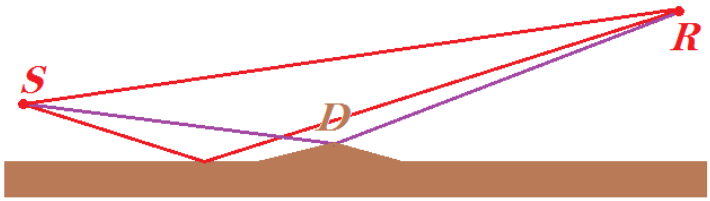
Note also that the point B might be located below the line of sight from the equivalent source and the equivalent receiver. In that case, a diffraction effect will automatically be accounted for by the evaluation of the excess attenuation in the unfolded propagation plane.

E.5 Raleigh's criterion

All three propagation models consider that diffraction has to be taken into account even if the diffracting edge is located below (but near to) the straight line connecting the source and the receiver. This may lead to incorrect predictions in case a low obstacle is placed on an almost flat ground.

Neither the ISO-9613 nor the JRC-2012 method specifies a criterion to determine whether a low obstacle has a significant diffraction effect or not. In order to overcome this shortcoming, the following solution, based on the insights gained during the Harmonoise and Imagine projects, has been introduced in the software.

Consider a low obstacle placed on an otherwise flat ground, as illustrated below:

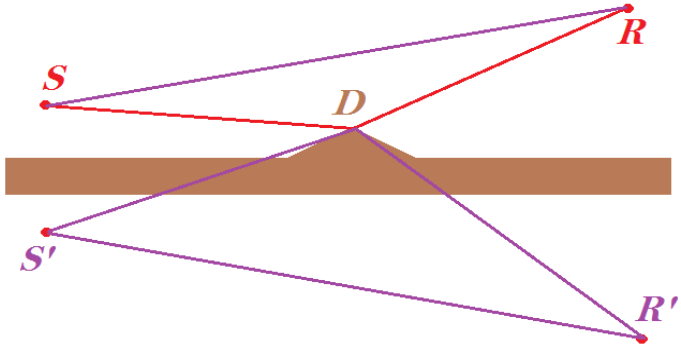


Raleigh's criterion states that such an obstacle has negligible effect on the reflection if the path difference for the scattered (or diffracted) wave is small compared to the path difference of the specular reflected wave:

$$\delta_D = |SD| + |DR| - |SR|$$

$$\Delta\delta = \delta_{spec} - \delta_D < \frac{\lambda}{8}$$

Equivalently, one may consider the following construct:



And rewrite Raleigh's criterion as following:

$$\delta_D = |SD| + |DR| - |SR|$$

$$\delta'_D = |S'D| + |DR'| - |S'R'|$$

$$\Delta\delta = \delta'_D - \delta_D < \lambda/4$$

This formulation immediately shows that any obstacle of almost zero height produces no significant diffraction effect, no matter its position in between the source (because $\delta'_D = \delta_D$ for any position D on flat ground). Moreover, it provides a deeper physical insight into the problem: an obstacle will provide a significant diffraction effect if it blocks the development of the surface wave (i.e. the destructive interference due to the complex interaction of the direct and the ground-reflected wave) which is responsible for the high attenuation of sound propagating over soft ground.

The revised formulation is used in both the implementations of ISO-9613-2 and JRC-2012.

For the ISO-9613-2 method, the relevant formulas become:

$$if\ \left(\Delta\delta = \delta'_D - \delta_D < \frac{\lambda}{4}\right)$$

$$A = A_{ground}$$

$$else$$
$$A = \max(A_{dif}, A_{ground})$$
$$A_{dif} = D_z + \min(0, A_{ground})$$

Because the ISO method does not explicitly models the shape of the terrain, image sources and receivers are simple created by considering their mirror positions with respect to their footprints on the local ground, i.e. $S(d_S, z_S + h_S) \rightarrow S(d_S, z_S - h_S)$.

For the JRC-2012 method, the relevant formulas are:

$$if\ \left(\Delta\delta = \delta'_D - \delta_D < \frac{\lambda}{4}\ and\ \delta_D < \frac{\lambda}{20}\right)$$
$$A = \Delta L_{ground}(S, R)$$
$$else$$
$$A = \Delta L_{ground}(S, D) + \Delta L_{dif}(S, D, R) + \Delta L_{ground}(D, R)$$

In this case, the image source and receiver are explicitly constructed by taken their mirror positions with respect to the mean planes calculated on either side of the potentially diffraction edge.

For both methods, the criterion is implemented in a local "*getGroundOrDiffraction*" method. Note that in case of the JRC-2012 model, this method is always called under favourable propagation conditions, even if in case the direct line between the source and the receiver is blocked by the diffracting obstacle. This is required because, due to the curving of the ray paths, a "blocked" path under homogeneous conditions may be transformed into an "unblocked" path under favourable condition (see 2nd case of figure VI.10 of the JRC-2012 report).

E.6 Complex transition function

In the Harmonoise propagation model, the ground effect is evaluated by means of the Chien-Soroka approximation, published in 1975[1]. As the details of this calculation are not given in the JRC draft report from 2010 or in the relevant background documents for the Harmonoise model, a brief description of the calculation scheme is provided here.

The sound pressure of a unit source placed above an infinite flat plane with an acoustical impedance Z is given by:

$$p = \frac{e^{jkr}}{4\pi r} + Q\frac{e^{jkr"}}{4\pi r"}$$

Where the spherical reflection coefficient is given by:

$$Q = R + (1 - R).F(p)$$

$$R = \frac{\cos\theta - \beta}{\cos\theta + \beta}$$

$$\beta = \frac{\rho_0 C}{Z}$$

$$F(z) = 1 + j\sqrt{\pi} z e^{-z^2} erfc(-jz)$$

$$z = \sqrt{\frac{jkr''}{2}} (\beta + \cos\theta)$$

The evaluation of the spherical reflection coefficient is based on the numerical evaluation of the auxiliary function:

$$w(z) = e^{-z^2} erfc(-jz)$$

$$z = x + j.y$$

The numerical evaluation of this function is carried out according to a note published by Chien and Soroka in 1979 [2]. The main difficulty in implementing the formula from this note resides in the correct interpretation of equation (7), i.e. for large negative values of either x or y. Although the note states the conditions "for $x < 0$ or $y < 0$", the actual formulas are given assuming $x > 0$ and $y > 0$, which is at the least creating confusion.

If $|x| \leq 3.9$ and $|y| \leq 3$ the function is evaluated using the series development, equation (8). As suggested by Chien and Soroka, the function is evaluated for $h = 0.8$ using up to the 5th term of the infinite series, so that the error remains bounded by $|E(h)| \leq 10^{-6}$. This approximation is valid, both for positive and negative values of x and y.

$$w(z) = H(y, x) + iK(y, x), \qquad (8)$$

Where:

$$H(y,x) = \frac{hy}{\pi^2(y^2+x^2)} + \frac{2yh}{\pi}\sum_{n=1}^{\infty}\frac{e^{-n^2h^2}(y^2+x^2+n^2h^2)}{(y^2-x^2+n^2h^2)^2+4y^2x^2} - \frac{y}{\pi}E(h)$$
$$+ P_2 \quad \text{if } y < \pi/h,$$
$$\text{or } +\tfrac{1}{2}P_2 \quad \text{if } y = \pi/h,$$
$$\text{or } -0 \quad \text{if } y > \pi/h,$$

$$K(y,x) = \frac{hx}{\pi(y^2+x^2)} + \frac{2xh}{\pi}\sum_{n=1}^{\infty}\frac{e^{-n^2h^2}(y^2+x^2-n^2h^2)}{(y^2-x^2+n^2h^2)^2+4y^2x^2} + \frac{x}{\pi}E(h)$$
$$-Q_2 \quad \text{if } y < \pi/h,$$
$$\text{or } -\tfrac{1}{2}Q_2 \quad \text{if } y = \pi/h,$$
$$\text{or } -0 \quad \text{if } y > \pi/h,$$

$$P_2 = 2e^{-(x^2+(2y\pi/h)-y^2)}\left[\frac{A_1C_1 - B_1D_1}{C_1^2+D_1^2}\right], \quad Q_2 = 2e^{-(x^2+(2y\pi/h)-y^2)}\left[\frac{A_1D_1 + B_1C_1}{C_1^2+D_1^2}\right],$$

$$A_1 = \cos 2xy, \quad B_1 = \sin 2xy, \quad C_1 = e^{-2y\pi/h} - \cos 2x\pi/h, \quad D_1 = \sin 2x\pi/h.$$

If $|x| > 3.9$ or $|y| > 3$ the function is evaluated using the rational approximations, equations (5) and (6). In order to avoid confusion, the actual implementation of the Harmonoise model relies on a secondary variable z" defined as

$$z" = |x| + j.|y|$$

The algorithm first calculates the value of $w" = w(z") = w(|x| + j.|y|)$, then uses the symmetry relations to calculate the values in case either or both $x < 0$ or $y < 0$.

Equation (6) is used in the case $|x| > 6$ or $|y| > 6$:

$$w(z) = iz\left(\frac{0{\cdot}5124242}{z^2 - 0{\cdot}2752551} + \frac{0{\cdot}05176536}{z^2 - 2{\cdot}724745}\right) + \eta(z), \qquad (6)$$

Equation (5) otherwise, i.e. for the ranges $3.9 \leq |x| < 6\,; 0 \leq |y| < 6$ or $0 \leq |x| < 6\,; 3 \leq |y| < 6$:

$$w(z) = iz\left(\frac{0{\cdot}4613135}{z^2 - 0{\cdot}1901635} + \frac{0{\cdot}09999216}{z^2 - 1{\cdot}7844927} + \frac{0{\cdot}002883894}{z^2 - 5{\cdot}5253437}\right) + \varepsilon(z), \qquad (5)$$

In case $x < 0$ or $y < 0$ the value of $w(z)$ is evaluated from the symmetry relations given in equation (7). Avoiding the confusing notations of the original paper, these relations may be rewritten as[10]:

$$\text{if} < 0 : \; w(x + j.y) = \overline{w(-x + j.y)} = \overline{w(|x| + j.y)}$$

$$\text{if} < 0 : \; w(x + j.y) = 2.e^{y^2 - x^2}(\cos 2xy - j \sin 2xy) - \overline{w(x - j.y)}$$

In case both $x < 0$ and $y < 0$, the above transformations are used successively to obtain:

$$w(|x| + j.|y|) \rightarrow w(x + j.|y|) \rightarrow w(x + j.y)$$

In case of doubt, please consult the C++ implementation of the algorithm which is listed below. This implementation of the Chien-Soroka formulas has been validated by numerical comparisons with alternative versions e.g. from Erik Salomons (TNO, Delft) and Gunnar Taraldsen (SINTEF, Trondheim).

```
Complex Fref (Complex z)
{
   Complex  j(0,1) ;
   Complex  erf ;

// first we calculate w(z) = exp(-z^2) * erfc(-j*z) for (x > 0, y > 0)

   double  x  = z.real() ;
   double  y  = z.imag() ;

// approximations for large values of the arguments
```

[10] Note that the sign in front of the sin(2xy) term is different from the original paper as we now consider y < 0 whereas the note of Chien and Soroka assumes y > 0 in equation (7), in contradiction with the text just above.

```
        if (fabs(x) > 3.9 || fabs(y) > 3)
        {
           Complex z1 = Complex (fabs(x), fabs(y));
           Complex z2 = z1 * z1 ;

           if ( fabs(x) > 6. || fabs(y) > 6. )
           {
               erf = j * z1 * (0.5124242
                   / (z2 - 0.2752551) + 0.05176536 / (z2 - 2.724745)) ;
           }
           else
           {
               erf = j * z1 * (0.461313500 / (z2 - 0.1901635)
                             + 0.099992160 / (z2 - 1.7844927)
                             + 0.002883894 / (z2 - 5.5253437)) ;
           }

        // careful about the signs here !

           if (x < 0) erf = conj(erf) ;
           if (y < 0) erf = 2 * exp(y*y-x*x)
                              * Complex(cos(2*x*y), -sin(2*x*y)) - conj(erf) ;
        }

     // series development for small values of x and y

        else
        {
           double h  = 0.8;
           double a1 = cos (2*x*y);
           double b1 = sin (2*x*y);
           double c1 = exp (-2*y*PI/h) - cos(2*x*PI/h);
           double d1 = sin (2*x*PI/h);
           double cd = c1*c1 + d1*d1;
           double p2,q2 ;

           if (cd == 0)
           {
              p2 = q2 = 1;
           }
           else
           {
              p2 = 2 * exp (-1*(x*x+2*y*PI/h-y*y))*(a1*c1-b1*d1) / cd;
              q2 = 2 * exp (-1*(x*x+2*y*PI/h-y*y))*(a1*d1+b1*c1) / cd;
           }

           double eh = 0.000001 ;
           double h1 = 0 ;
           double h2 = 0 ;
           for (int n = 1 ; n <= 5 ; n++)
           {
              double x1 = (y*y+x*x+n*n*h*h);
              double x2 = (y*y-x*x+n*n*h*h);
              double x3 = (y*y+x*x-n*n*h*h);
              h1 += exp(-1*(n*n)*h*h)*x1/(x2*x2+4*y*y*x*x);
              h2 += exp(-1*(n*n)*h*h)*x3/(x2*x2+4*y*y*x*x);
```

```
            }

            double hyx = h*y / (PI*(x*x+y*y)) + 2*y*h*h1/PI - y*eh/PI;
            double kyx = h*x / (PI*(x*x+y*y)) + 2*x*h*h2/PI + x*eh/PI;

            if (y < PI/h)
            {
               hyx = hyx + p2;
               kyx = kyx - q2;
            }

            if (y == PI/h)
            {
               hyx = hyx + 0.5 * p2;
               kyx = kyx - 0.5 * q2;
            }

            erf = Complex (hyx, kyx) ;
         }

      // now return Fref(w) = 1 + j * sqrt(PI) * z * [exp(-z^2) * erfc(-j*z)]

         return 1. + j * sqrt(PI) * z * erf ;
      }
```

References:

[1] Ingard U, Lamb G. Jr, "Effect of Reflecting Plane on the Power Output of Sound Sources", J. Acoust. Soc. Am., 29:743-744 (1957)

[2] ISO 354, 2003, Acoustics: Measurement of sound absorption in a reverberation room.

[3] EN 1793-1, 1997, Road traffic noise reduction devices – Test method for determining the acoustical performances. Part 1 – Intrinsic characteristics of sound absorption.

[4] Delanay & Bazley, "Acoustical properties of fibrous absorbing materials", Applied Acoustics 3 (1970), p.150-116.

[5] Morse & Ingard, "Theoretical Acoustics", McGraw-Hill (1968), eq. 9.5.8 page 580.

[6] NF S 31-133, Bruit dans l'environnement - Calcul de niveaux sonores (février 2011).

[7] J. Deygout, " Les Cours de L'Ecole Supérieure d'Electricité - Données fondamentales de la propagation radioélectrique", Editions Eyrolles, 1994.

[8] C.F. Chien and W.W. Soroka. Sound propagation along an impedance plane. Journal of Sound and Vibration (1975), **43**, 9-20.

[9] C.F. Chien and W.W. Soroka. A note on the calculation of sound propagation along an impedance surface. Journal of Sound and Vibration (1980) **69**(2), 340-343.