NOISEMODELLING PRESENTATION

1 Numerical Model 3
1.1 Emission Numerical Model .................................................. 3
1.2 Ray Tracing ......................................................................... 3
1.3 Propagation Numerical Model .............................................. 3

2 Validation .............................................................................. 5

3 Some contributions using NoiseModelling 7
3.1 Standard Noise maps ............................................................... 7
3.2 Dynamic Noise maps ............................................................... 7
3.3 Probabilistic & Multi-sources Noise maps .............................. 8
3.4 Sensitivity Analysis & data assimilation ................................. 8
3.5 Auralisation ......................................................................... 9

4 Get Started - Tutorial 11
4.1 Requirements: Install Java and set JAVA_HOME environment variable ......................................................... 11
4.2 Step 1: Download the latest release ....................................... 11
4.3 Step 2: Run GeoServer ........................................................... 12
4.4 Step 3: Run WPSBuilder ......................................................... 12
4.5 Step 4: Upload files to database ............................................. 12
4.6 Step 5: Run Calculation .......................................................... 13
4.7 Step 6: Export (& see) the results .......................................... 14
4.8 Step 7: Know the possibilities ................................................. 15

5 Noise Map from OSM - Tutorial 17
5.1 Prerequisites ........................................................................ 17
5.2 Step 1: Importing OSM data to the database ......................... 17
5.3 Step 2: Viewing tables and data layers .................................. 18
5.4 Step 3: Generating a Receiver table ...................................... 19
5.5 Step 4: Using Noise Modelling .............................................. 19
5.6 Step 5: Create Isosurfaces map ............................................. 19
5.7 Step 6: Viewing the result ...................................................... 19

6 Tutorials - FAQ 21
6.1 Shapefiles ? GeoJSON ? ........................................................ 21
6.2 PostgreSQL ? H2 ? ............................................................... 21
6.3 OSM ? .................................................................................. 22
6.4 Metric SRID ? ..................................................................... 22

7 WPS Blocks 23
7.1 WPS general presentation ..................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>NoiseModelling and WPS</td>
<td>23</td>
</tr>
<tr>
<td>7.3</td>
<td>Create your own WPS block</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>WPS Builder</td>
<td>25</td>
</tr>
<tr>
<td>8.1</td>
<td>What is WPS Builder?</td>
<td>25</td>
</tr>
<tr>
<td>8.2</td>
<td>What is the color of the block in WPSBuilder?</td>
<td>25</td>
</tr>
<tr>
<td>8.3</td>
<td>Can I save my WPSBuilder project?</td>
<td>25</td>
</tr>
<tr>
<td>8.4</td>
<td>Why everything is wrong when I use &quot;enter&quot;?</td>
<td>25</td>
</tr>
<tr>
<td>8.5</td>
<td>I can’t link process block between them?</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>Create your own WPS block</td>
<td>27</td>
</tr>
<tr>
<td>9.1</td>
<td>Presentation</td>
<td>27</td>
</tr>
<tr>
<td>9.2</td>
<td>General Structure</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>Manipulate your database with dBeaver</td>
<td>29</td>
</tr>
<tr>
<td>10.1</td>
<td>Presentation</td>
<td>29</td>
</tr>
<tr>
<td>10.2</td>
<td>Download dBeaver</td>
<td>29</td>
</tr>
<tr>
<td>10.3</td>
<td>Connect dBeaver to your database</td>
<td>29</td>
</tr>
<tr>
<td>10.4</td>
<td>Use dBeaver</td>
<td>29</td>
</tr>
<tr>
<td>10.5</td>
<td>Connect dBeaver to NoiseModelling libraries</td>
<td>30</td>
</tr>
<tr>
<td>11</td>
<td>Use NoiseModelling by scripting</td>
<td>31</td>
</tr>
<tr>
<td>11.1</td>
<td>Introduction</td>
<td>31</td>
</tr>
<tr>
<td>11.2</td>
<td>The Web Processing Service API</td>
<td>31</td>
</tr>
<tr>
<td>11.3</td>
<td>How it works</td>
<td>31</td>
</tr>
<tr>
<td>11.4</td>
<td>Method to easily find inputs query</td>
<td>31</td>
</tr>
<tr>
<td>11.5</td>
<td>Get started using Python</td>
<td>34</td>
</tr>
<tr>
<td>12</td>
<td>Use NoiseModelling with a PostGIS database</td>
<td>37</td>
</tr>
<tr>
<td>12.1</td>
<td>Introduction</td>
<td>37</td>
</tr>
<tr>
<td>12.2</td>
<td>Connect with Java</td>
<td>37</td>
</tr>
<tr>
<td>13</td>
<td>Get Started</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>Support</td>
<td>45</td>
</tr>
<tr>
<td>15</td>
<td>License</td>
<td>47</td>
</tr>
<tr>
<td>16</td>
<td>Indices and tables</td>
<td>49</td>
</tr>
</tbody>
</table>
This is the official NoiseModelling User Guide.

NoiseModelling is a library capable of producing noise maps of cities. This tool is almost compliant with the CNOSSOS-EU standard method for the noise emission (only road traffic) and noise propagation. It can be freely used either for research and education, as well as by experts in a professional use.

This plugin is distributed under GPL 3 license and is developed by the DECIDE team of the Lab-STICC (CNRS) and by the Mixt Research Unit in Environmental Acoustics UMRAE (Ifsttar).

- for more information on NoiseModelling, visit the official NoiseModelling website
- to contribute to NoiseModelling from the source code, follow the instructions
- to contact the development team, use the email contact@noise-planet.org or let an issue: https://github.com/Ifsttar/NoiseModelling/issues

Cite as: Erwan Bocher, Gwenaël Guillaume, Judicaël Picaud, Gwendall Petit, Nicolas Fortin. NoiseModelling: An Open Source GIS Based Tool to Produce Environmental Noise Maps. ISPRS International Journal of Geo-Information, MDPI, 2019, 8 (3), pp.130. 10.3390/ijgi8030130. hal-02057736

Fundings:

Research projects:
- ANR Eval-PDU (ANR-08-VILL-0005) 2008-2011
- ANR CENSE (ANR-16-CE22-0012) 2017-2021
- the Nature4cities (N4C) project, funded by European Union’s Horizon 2020 research and innovation programme under grant agreement No 730468

Institutional (public) fundings:
- Univ Gustave Eiffel (formerly Ifsttar, formerly LCPC), CNRS, UBS, ECN, Cerema

Private fundings:
- Airbus Urban Mobility

Note:
• The official documentation is available in English only.
• Some illustrations may refer to previous versions of NoiseModelling.
• If you observe some mistakes or errors, please contact us at contact@noise-planet.org or let an issue here.
• You can also contribute to the documentation
1.1 Emission Numerical Model

1.1.1 Traffic emission model

The emission model of the implemented traffic is the CNOSSOS-EU model.

Note: Current model includes the emission coefficients a and b presented in the report “Amendments for CNOSSOS-EU” (Kok, 2019)

1.1.2 Other emission models

Other emission models are not included within the release 3.0.

1.2 Ray Tracing

The ray tracing algorithm is a rubber-band like algorithm as specified in CNOSSOS-EU.

Warning:
- rays backwards to the source or receiver are not taken into account. For example, if a receiver is located inside a U-shaped building, only diffractions on horizontal edges will be taken into account.

1.3 Propagation Numerical Model

The emission model of the implemented traffic is the CNOSSOS-EU model.

Warning:
- the rays under favorable conditions are subject to questioning. The current version is not final.
- the Rayleigh criterion is subject to questioning. The current version does not integrate calculation that involve this criterion.
• taking into account 15 degrees obstacles are subject to questioning. The current version doesn’t integrate calculation that involve 15 degrees obstacles.
For numerical model validation, please refer to CNOSSOS-EU papers, this is independant to NoiseModelling.

For implementation validation, a large set of unit tests are present in the code. Please consult them.
SOME CONTRIBUTIONS USING NOISEMODELLING

3.1 Standard Noise maps


3.2 Dynamic Noise maps


https://www.youtube.com/watch?v=jl8tASDr-uQ&t=133s
3.3 Probabilistic & Multi-sources Noise maps

- AUMOND P., CAN A. Probabilistic modeling framework to predict traffic sound distribution, Proceedings of Euronoise, Hersonissos, Crete, 27-31 May 2018

3.4 Sensitivity Analysis & data assimilation


https://www.youtube.com/watch?v=orc5ZbN2dlY
3.5 Auralisation

- F. ROHRLICH, C. VERRON (Noise Makers), Captation et Simulation d’Ambiances Urbaines Spatialisées , 2018-2019

![Diagram of Auralisation process]

**Figure 24** : Schéma de principe du prototype de simulateur d’ambiances urbaines. À gauche, les éléments liés à NoiseModelling, qui sont précalculés; à droite, les éléments temps réel (Unity et Max/MSP).
4.1 Requirements: Install Java and set JAVA_HOME environment variable

Please install JAVA version v8.x. Currently only version 8 of Java is compatible.

- Download Java here: https://www.java.com/fr/download/
- You can check if JAVA_HOME environment variable is well settled to your last installed java folder using `echo %JAVA_HOME%` (windows) or `echo $JAVA_HOME` (linux) in your command prompt. You should have a result similar to `C:\Program Files (x86)\Java\jre1.8.x\`
- If you don’t have this result, it is probably because your JAVA_HOME environment variable is not well settled. To set you JAVA_HOME environment variable you can adapt (with the JAVA version number) you installed and use the following command line: `setx JAVA_HOME "C:\Program Files (x86)\Java\jre.1.8.x\` in your command prompt. You can also refer to this document for example.
- You may have to reboot your command prompt after using the precedent command line before printing again `echo %JAVA_HOME%` (windows) or `echo $JAVA_HOME` (linux).

**Warning:** The command prompt should print `C:\Program Files (x86)\Java\jre1.8.x\` without the bin directory. If JAVA_HOME is settled as `C:\Program Files (x86)\Java\jre1.8.x\bin`, it will not work. It should also point to a JRE (Java Runtime Environment) Java environment and not JDK.

4.2 Step 1: Download the latest release

- Download the latest release on Github.
- Unzip the downloaded file into a chosen directory.

**Note:**
- Only from version 3.3, NoiseModelling releases include the user interface described in this tutorial.
- The chosen directory can be anywhere but be sure that you have write access. If you are using the computer of your company, the Program Files folder is probably not a good idea.
4.3 Step 2: Run GeoServer

NoiseModelling connects to a PostGIS or H2GIS database. The database needs to be hosted by a server. In this tutorial the server type is GeoServer and the database type is H2GIS. Those tools are included in the archive.

To run the server, please execute “startup” from your own Geoserver folder:

- NoiseModelling\bin\startup.bat for Windows Users
- NoiseModelling\bin\startup.sh for Linux Users (check authorize file execution in property of this file before)

and wait until INFO:oejs.Server:main:Started is written in your command prompt.

**Warning:** The server launch can take some time. Be patient.

Your local server is now started.

**Warning:** Your server will be open as long as the command window is open. If you close it, the server will automatically be closed and you will not be able to continue with the tutorial.

**Tip:** You can consult it via your web browser: http://localhost:9580/geoserver/web/

- **login** (default): admin
- **password** (default): admin

**Warning:** On older versions, the url was: http://localhost:8080/geoserver/web/

4.4 Step 3: Run WPSBuilder

The WPSBuilder is the user interface used to communicate between the GeoServer and NoiseModelling.

To launch WPSBuilder, go to http://localhost:9580 using your preferred web browser.

4.5 Step 4: Upload files to database

To compute your first noise map, you will need 5 layers: Buildings, Roads, Ground type, Topography (DEM) and Receivers.

In the noisemodelling\data_dir\data\wpsdata folder, you will find 5 files (4 shapefiles and 1 geojson) corresponding to these layers.

You can import these layers in your database using the Import File or Import Folder blocks.

- Drag **Import File** block into Builder window
- Select **Path of the input File** block and type data_dir\data\wpsdata\buildings.shp in the field **pathFile**
- Then click on Run Process after selecting one of the sub-blocks of your process
Files are uploaded to database when the Console window displays The table x has been uploaded to database.

Repeat this operation for other files:

- data_dir/data/wpsdata/ground_type.shp
- data_dir/data/wpsdata/receivers.shp
- data_dir/data/wpsdata/ROADS2.shp
- data_dir/data/wpsdata/dem.geojson

Note: You can find all files in your own NoiseModelling folder, at direction /data_dir/data/wpsdata/

Note:
- if you have the message Error opening database, please refer to the note in Step 1.
- The process is supposed to be quick (<5 sec.). In case of out of time, try to restart the Geoserver (see Step 2).
- Orange blocks are mandatory
- Beige blocks are optional
- if all input blocks are optional, you must modify at least one of these blocks to be able to run the process
- Blocks get solid border when they are ready to run
- Find here more information about WPS Builder.

4.6 Step 5: Run Calculation

To run Calculation you have to drag the block Noise_level_from_traffic into WPS Builder window.

Then, select the orange blocks and indicate the name of the corresponding table your database, for example:

- Building table name: BUILDINGS
- Sources table name: ROADS2
- Receivers table name: RECEIVERS

Then, you can run the process.

The tables LDAY_GEOM, LEVENING_GEOM, LNIGHT_GEOM and LDEN_GEOM will be created in your database.

Note: If you want to know more about the format of the input tables, you can refer to the WPS Blocks section.

Tip: If you want you can try to change the different parameters.
4.7 Step 6: Export (& see) the results

You can now export the output table in your favorite export format using Export Table block giving the path of the file you want to create (including its extension, for example: c:/home/receivers.geojson).

For example, you can choose to export the table in shp format. This format can be read with many GIS tools such as the open source softwares QGIS and SAGA.

To obtain the following image, use the styling vector options in your GIS and assign a color gradient to LAEQ column of your exported LDAY_GEOM table.
4.8 Step 7: Know the possibilities

Now that you have finished this first step, take the time to read the description of each of the WPS blocks present in your version of NoiseModelling.

By clicking on each of the inputs or outputs, you can also get additional information.
NOISE MAP FROM OSM - TUTORIAL

5.1 Prerequisites

- You need at least NoiseModelling v.3.0.6, the best is always to use last release.
- If you have just finished the first tutorial, please clean your database with the WPS block `Clean_Database`

**Warning:** Don’t forget to check the *Are you sure* check box before running the process.

5.2 Step 1: Importing OSM data to the database

5.2.1 Exporting data from openstreetmap.org

- Go to https://www.openstreetmap.org
- Zoom in on the area you want to export
- Export the zone in .osm or .osm.gz format with *Export* button

**Warning:** For Mac users, safari may automatically rename the file to “map.osm.xml”. Simply delete the “.xml” extension before import.

5.2.2 Import to the database

- Use the WPS block `Import_OSM`

**Note:** About the Coordinate System (EPSG code)

In several input files, you need to specify coordinates, e.g road network. You can’t use the WGS84 coordinates (i.e. GPS coordinates). Acoustic propagation formulas make the assumption that coordinates are metric. Many countries and regions have custom coordinate system defined, optimized for usages in their appropriate areas. It might be best to ask some GIS specialists in your region of interest what the most commonly used local coordinate system is and use that as well for your data. If you don’t have any clue about what coordinate system is used in your region, it might be best to use the Universal Transverse Mercator coordinate system. This coordinate system divides the world into multiple bands, each six degrees width and separated into a northern and southern part, which is called UTM zones (see [http://en.wikipedia.org/wiki/UTM_zones#UTM_zone](http://en.wikipedia.org/wiki/UTM_zones#UTM_zone) for more details). For each zone, an optimized coordinate
system is defined. Choose the UTM zone which covers your region (Wikipedia has a nice map showing the zones) and use its coordinate system. 

Here is the map: https://upload.wikimedia.org/wikipedia/commons/e/ed/Utm-zones.jpg

---

Note: 
- Inform the target projection identifier field with the corresponding SRID 
- Enter the path to the file map.osm 
- Select OsmToInputData box then click on the green button

---

**Warning:** The current import script from open street map can produce geometries incompatible with NoiseModelling. If an area is a problem try to reduce the area. A much more robust version of this script will be available soon.

---

Three tables must be created: GROUND, BUILDINGS, ROADS

### 5.3 Step 2: Viewing tables and data layers

#### 5.3.1 Using WPSBuilder

- The contents of the database can be viewed using *Display Database*.
- A spatial layer can be visualized using *Table_Visualization_Map*.
- A data table can be visualized using *Table_Visualization_Data*.

#### 5.3.2 Viewing the database

- **Export a table**

It is also possible to export the tables via *Export_Table* in Shapefile, CSV or GeoJSON format.

- **Viewing a table**

Then import these tables into your preferred Geographic Information System (e.g. OrbisGIS, QGIS). You can then graphically visualize your data layer, but also the data it contains. Take the time to familiarize yourself with your chosen GIS.

- **Adding a background map**

OrbisGIS/QGIS allow you to add an OSM background map: https://wiki.openstreetmap.org/wiki/QGIS

- **Change colors**

OrbisGIS/QGIS allow you to change layer colors (e.g. Surface_osm in green, Buildings_OSM in gray, ROADS in red).
5.4 Step 3: Generating a Receiver table

The locations of noise level evaluation points needs to be defined.

Use Delaunay_Grid with the previously generated BUILDINGS table as the buildings table, and ROADS as Sources table name. Other parameters are optional.

Don’t forget to view your resulting layer in WPSBuilder or OrbisGIS/QGIS to check that it meets your expectations. This processing block will give the possibility to generate a noise map later.

5.5 Step 4: Using Noise Modelling

5.5.1 Associating an emission noise level with roads

The Road_Emission_from_Traffic block is used to generate a road layer, called LW_ROADS, containing LW emission noise level values in accordance with the emission laws of the CNOSSOS model. The format of the input road layer can be found in the description of the WPS Block.

Don’t forget to view your resulting layer in WPSBuilder or OrbisGIS/QGIS to verify that it meets your expectations.

5.5.2 Source to Receiver Propagation

The Noise_level_from_source block allows to generate a layer of receiver points with associated sound levels corresponding to the sound level emitted by the sources (created table LW_ROADS) propagated to the receivers according to the CNOSSOS propagation laws.

5.6 Step 5: Create Isosurfaces map

Create an interpolation of levels between receivers points using the block Create_Isosurface.

Set LDEN_GEOM as Name of the noise table.

5.7 Step 6: Viewing the result

5.7.1 Exporting

You can then export the output table CONTOURING_NOISE_MAP via Export_Table in shapefile or GeoJSON format.
5.7.2 Viewing

You can view this layer in your favorite GIS. You can then apply a color gradient on ISOLVL field, the noise level intervals are in ISOLABEL field.
6.1 Shapefiles ? GeoJSON ?

**Shapefile** is a file format for geographic information systems (GIS).

Its extension is classically SHP, and it is always accompanied by two other files with the same name and extensions:

- DBF, a file that contains attribute data relating to the objects contained in the shapefile,
- SHX, file that stores the index of the geometry.

Other files can also be provided:

- prj - coordinate system information, using the WKT (Well Known Text) format;
- sbn and sbx - spatial shape index;
- fbn and fbx - spatial shape index for read-only shapefiles;
- ain and aih - attribute index for active fields in a table or in a theme attribute table;
- etc.

**GeoJSON** (Geographic JSON) is an open format for encoding simple geospatial data sets using the JSON (JavaScript Object Notation) standard. It is an alternative to the Shapefile format. It has the advantage of being readable directly in a text editor.

6.2 PostGreSQL ? H2 ?

**PostGreSQL** & **H2** are two database management system (DBMS). They are used to store, manipulate or manage, and share information in a database, ensuring the quality, permanence and confidentiality of the information, while hiding the complexity of the operations. NoiseModelling can connect to DBMS in H2 - H2GIS or PostGreSQL - PostGIS format.
6.3 OSM

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world. The geodata underlying the map is considered the primary output of the project. The creation and growth of OSM has been motivated by restrictions on use or availability of map data across much of the world, and the advent of inexpensive portable satellite navigation devices. OSM is considered a prominent example of volunteered geographic information.

6.4 Metric SRID

Spatial reference systems can be referred to using a SRID integer, including EPSG codes.

In several input files, you need to specify coordinates, e.g. road network. It is strongly suggested not to use WGS84 coordinates (i.e. GPS coordinates). Acoustic propagation formulas make the assumption that coordinates are metric. Many countries and regions have custom coordinate system defined, optimized for usages in their appropriate areas. It might be best to ask some GIS specialists in your region of interest what the most commonly used local coordinate system is and use that as well for your data. If you don’t have any clue about what coordinate system is used in your region, it might be best to use the Universal Transverse Mercator coordinate system. This coordinate system divides the world into multiple bands, each six degrees width and separated into a northern and southern part, which is called UTM zones (see http://en.wikipedia.org/wiki/UTM_zones#UTM_zone for more details). For each zone, an optimized coordinate system is defined. Choose the UTM zone which covers your region (Wikipedia has a nice map showing the zones) and use its coordinate system.

Here is the map: https://upload.wikimedia.org/wikipedia/commons/e/ed/Utm-zones.jpg

Note: We recommend using the website https://epsg.io/ to find the appropriate SRID code for your location.
7.1 WPS general presentation

The OGC Web Processing Service (WPS) Interface Standard provides rules for standardizing inputs and outputs (requests and responses) for invoking geospatial processing services, such as polygon overlay, as a web service.

The WPS standard defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and clients’ discovery of and binding to those processes.

7.2 NoiseModelling and WPS

NoiseModelling v3.0.0 comes with 7 WPS blocks. WPS scripts for GeoServer are written in groovy language.

They are located in Geoserver\data_dir\scripts\wps directory

**Tip:** To know the functionality of each WPS block, wait a few moments with your mouse on the block, a tooltip text will appear.

**Note:** With each new version, new blocks are added. Be curious and check out the latest version!

7.3 Create your own WPS block

Please see Advanced Users Section, because now you want to be one!
8.1 What is WPS Builder?

WPS Builder allows for the creating of graphical process workflows that can be easily executed and reproduced. It allows Web Processing Services to operate through a user interface.

We have developed a version of WPS Builder adapted to the needs of NoiseModelling. This version being very close to the original version, do not hesitate to consult the official documentation: WPS Builder documentation

8.2 What is the color of the block in WPSBuilder?

- Orange block are mandatory
- Beige blocks are optional
- Blocks get solid border when they are ready
- Green block are unfortunately useless

8.3 Can I save my WPSBuilder project?

You can also save your work by clicking on File icon.

8.4 Why everything is wrong when I use “enter”?

Don’t click on your enter keyboard key, it refreshes web page.

8.5 I can’t link process block between them?

It is normal... Still doesn’t work!
CREATE YOUR OWN WPS BLOCK

9.1 Presentation

The OGC Web Processing Service (WPS) Interface Standard provides rules for standardizing inputs and outputs (requests and responses) for invoking geospatial processing services as a web service.

WPS scripts for GeoServer are written in groovy language. They are located in the Geoserver\data_dir\scripts\wps directory.

To help you build your WPS block, you will find a template in the Geoserver\data_dir\scripts\template directory

Note: Don’t be shy, if you think your block can be useful to the community, you can redistribute it using github or by sending it directly to us.

Tip: The best way to make your own WPS is to be inspired by those that are already made. See how the tutorial is built or contact us for many more examples.

9.2 General Structure

9.2.1 1. Import used libraries

```java
import geoserver.GeoServer
import geoserver.catalog.Store
```

9.2.2 2. WPS Script meta data

```java
title = '....'
description = '......'
```
9.2.3 3.WPS Script input & output

```java
inputs = [
    inputparameter1: [name: '...', description : '...', title: '...', type: String.class],
    inputparameter2: [name: '...', description : '...', title: '...', type: String.class]
]

outputs = [
    outputparameter: [name: '...', title: '...', type: String.class]
]
```

9.2.4 4.Set connection method

```java
def static Connection openPostgreSQLDataStoreConnection() {
    Store store = new GeoServer().catalog.getStore("h2gisdb")
    JDBCDataStore jdbcDataStore = (JDBCDataStore)store.getDataStoreInfo().getDataStore(null)
    return jdbcDataStore.getDataSource().getConnection()
}
```

9.2.5 5.Set main method to execute

```java
def run(input) {

    // Open connection and close it at the end
    openPostgreSQLDataStoreConnection(dbName).withCloseable { Connection connection ->
        // Execute code here
        // for example, run SQL command lines
        Sql sql = new Sql(connection)
        sql.execute("drop table if exists TABLETODROP")
    }

    // print to Console windows
    return [result : 'Ok ! ']
}
```
MANIPULATE YOUR DATABASE WITH DBEAVER

10.1 Presentation

DBeaver is free universal SQL client/database tool for developers and database administrators. DBeaver is able to connect to H2GIS database which is the one used.

10.2 Download dBeaver

You can download dBeaver on the webpage

10.3 Connect dBeaver to your database

Note: Be sure that the geoserver is closed. It is not possible to connect dBeaver and GeoServer at the same time.

1. Run dBeaver
2. Add a new connection
3. If you use a h2gis type database please select ‘H2GIS embedded’
4. Browse your database. By default it is in Geoserver\data_dir and the name is h2gisdb.mv.db
5. Open it!

10.4 Use dBeaver

Now you can use the full potential of dBeaver and the h2gis database. You can explore, display and manage your database.

Many spatial processing are possible with H2GIS. Please see the H2GIS website.
10.5 Connect dBBeaver to NoiseModelling libraries

You have to load the NoiseModelling library using the grab annotation at the beginning of its script as here.
11.1 Introduction

You master the use of the “WPS Builder” user interface. Now you want to stop using this interface and run the processes with your favorite programming language.

Well know that it is possible!

It’s actually quite simple.

All you have to do is prepare the HTTP calls you will make to Geoserver.

11.2 The Web Processing Service API

The WPS is not something we did on our own. It’s a standardized interface very well established in the geospatial community. So you will find a lot of documentation and libraries.

You can first learn more about it:

https://en.wikipedia.org/wiki/Web_Processing_Service

11.3 How it works

Processing scripts are hosted by a GeoServer instance. This software contain a web server that listen for localhost requests.

What your scripts need to do is simply access this web server like a web browser would. The difficulty being the creation of the entries expected by the service.

11.4 Method to easily find inputs query

The POST request must contain the expected inputs.

You can use your navigator debug mode with the WPS Builder GUI In order to generate the inputs for you.

Once in WPS Gui press F12 key to enter debug mode, then click on network tab:

From here click on “Run process”, now you have the complete url and query to copy-paste into your script.
Chapter 11. Use NoiseModelling by scripting

Your script

Simple HTTP call with POST Method

1. POST ./geoserver/wps/

http response

2. run groovy code

3. process response

GeoServer

wps service

groovy wps block

{http..}
11.4. Method to easily find inputs query
11.5 Get started using Python

The following script runs the exact WPS blocks used in the Get Started tutorial.

We can use the String.Template class in order to substitute our inputs.

urllib.request module is also helpful for post queries.

```python
import urllib.request
from string import Template

import_file = Template('import_file = Template('<p0:Execute xmlns:p0="http://www.opengis.net/wps/1.0.0" service="WPS" version="1.0.0"><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">Import_and_Export:Import_File</p1:Identifier><p0:DataInputs><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">pathFile</p1:Identifier><p0:Data><p0:LiteralData>$path</p0:LiteralData></p0:Data></p0:Input></p0:DataInputs><p0:ResponseForm><p0:RawDataOutput><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">result</p1:Identifier></p0:RawDataOutput></p0:ResponseForm></p0:Execute>')

get_lday = Template('get_lday = Template('<p0:Execute xmlns:p0="http://www.opengis.net/wps/1.0.0" service="WPS" version="1.0.0"><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">NoiseModelling</p1:Identifier><p0:DataInputs><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">tableReceivers</p1:Identifier><p0:Data><p0:LiteralData>$table_receivers</p0:LiteralData></p0:Data></p0:Input><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">tableBuilding</p1:Identifier><p0:Data><p0:LiteralData>$table_buildings</p0:LiteralData></p0:Data></p0:Input><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">tableDEM</p1:Identifier><p0:Data><p0:LiteralData>$table_dem</p0:LiteralData></p0:Data></p0:Input><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">tableRoads</p1:Identifier><p0:Data><p0:LiteralData>$table_roads</p0:LiteralData></p0:Data></p0:Input></p0:DataInputs><p0:ResponseForm><p0:RawDataOutput><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">result</p1:Identifier></p0:RawDataOutput></p0:ResponseForm></p0:Execute>')

export_table = Template('export_table = Template('<p0:Execute xmlns:p0="http://www.opengis.net/wps/1.0.0" service="WPS" version="1.0.0"><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">Import_and_Export:Export_Table</p1:Identifier><p0:DataInputs><p0:Input><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">tableToExport</p1:Identifier><p0:Data><p0:LiteralData>$table_to_export</p0:LiteralData></p0:Data></p0:Input></p0:DataInputs><p0:ResponseForm><p0:RawDataOutput><p1:Identifier xmlns:p1="http://www.opengis.net/ows/1.1">result</p1:Identifier></p0:RawDataOutput></p0:ResponseForm></p0:Execute>')
```
def call_geoserver(data):
    req = urllib.request.Request(url='http://localhost:9580/geoserver/ows',
        data=bytes(data, encoding="utf8"),
        method='POST')
    req.add_header('Content-Type', 'application/xml; charset=utf-8')
    with urllib.request.urlopen(req) as f:
        print(f.status)
        print(f.reason)
        if f.status == 200:
            print(str(f.read(), encoding="utf8"))

call_geoserver(import_file.substitute({"path": "data_dir/data/wpsdata/buildings.shp"}))
call_geoserver(import_file.substitute({"path": "data_dir/data/wpsdata/ground_type.shp"}))
call_geoserver(import_file.substitute({"path": "data_dir/data/wpsdata/receivers.shp"}))
call_geoserver(import_file.substitute({"path": "data_dir/data/wpsdata/roads.shp"}))
call_geoserver(import_file.substitute({"path": "data_dir/data/wpsdata/dem.geojson"}))
call_geoserver(get_lday.substitute({"table_receivers": "RECEIVERS", "table_buildings": "BUILDINGS", "table_roads": "ROADS", "table_dem": "DEM"}))
call_geoserver(export_table.substitute({"table_to_export": "LDAY_GEOM", "export_path": "lday_geom.shp"}))
USE NOISEMODELLING WITH A POSTGIS DATABASE

12.1 Introduction

NoiseModelling is distributed with the GeoServer (http://geoserver.org/) application. This application has been pre-configured to use H2GIS as the default database.

H2GIS does not need to be configured or installed on the system and is therefore perfectly suitable as a default database.

However, this database is less efficient than the Postgre/PostGIS database, which has a larger community of contributors/users.

NoiseModelling is written with the idea of maintaining H2GIS/PostGIS compatibility.

This tutorial will not cover the steps for installing and configuring a PostGIS database.

12.2 Connect with Java

First you have to add some libraries. We will use PostgreSQL/PostGIS wrapper available in the H2GIS library:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <properties>
        <h2gis-version>1.5.1-SNAPSHOT</h2gis-version>
        <noisemodelling-version>3.0.1-SNAPSHOT</noisemodelling-version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.12</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>noisemodelling-emission</artifactId>
            <version>${noisemodelling-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>noisemodelling-propagation</artifactId>
        </dependency>
    </dependencies>
</project>
```
The new dependency here is postgis-jts-osgi. It contains some code to convert PostGIS geometries objects into/from JTS objects.

In your code you have to import the PostGIS wrapper class and some utility class:

```java
import org.h2gis.functions.io.geojson.GeoJsonRead;
import org.h2gis.postgis_jts_osgi.DataSourceFactoryImpl;
import java.net.ConnectException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashSet;
import java.util.Locale;
```

Then use it to connect to you local or remote PostGIS database and obtain a valid JDBC connection object:

```java
public static void main() throws Exception {
    DataSourceFactoryImpl dataSourceFactory = new DataSourceFactoryImpl();
    Properties p = new Properties();
    p.setProperty("serverName", "localhost");
    p.setProperty("portNumber", "5432");
    p.setProperty("databaseName", "postgres");
    p.setProperty("user", "postgres");
    p.setProperty("password", "");
    try (Connection connection = SFSUtilities.wrapConnection(dataSourceFactory.
        createDataSource(p).getConnection())) {
        Statement sql = connection.createStatement();
    }
}
```

Finally you can use the NoiseModelling functions as usual:
package org.noise_planet.nmtutorial01;

import org.h2gis.api.EmptyProgressVisitor;
import org.h2gis.api.ProgressVisitor;
import org.h2gis.functions.io.csv.CSVDriverFunction;
import org.h2gis.functions.io.geojson.GeoJsonRead;
import org.h2gis.postgis_jts_osgi.DataSourceFactoryImpl;
import org.h2gis.utilities.SFSUtilities;
import org.junit.Test;
import org.noise_planet.noisemodelling.emission.jdbc.LDENConfig;
import org.noise_planet.noisemodelling.emission.jdbc.LDENPointNoiseMapFactory;
import org.noise_planet.noisemodelling.propagation.ComputeRaysOut;
import org.noise_planet.noisemodelling.propagation.IComputeRaysOut;
import org.noise_planet.noisemodelling.propagation.RootProgressVisitor;
import org.noise_planet.noisemodelling.propagation.jdbc.PointNoiseMap;
import org.postgresql.util.PSQLException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.net.ConnectException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.HashSet;
import java.util.Locale;
import java.util.Properties;
import java.util.Set;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class Main {
  static Logger LOGGER = LoggerFactory.getLogger(Main.class);

  public static void main() throws Exception {
    DataSourceFactoryImpl dataSourceFactory = new DataSourceFactoryImpl();
    Properties p = new Properties();
    p.setProperty("serverName", "localhost");
    p.setProperty("portNumber", "5432");
    p.setProperty("databaseName", "postgres");
    p.setProperty("user", "postgres");
    p.setProperty("password", "");
    try (Connection connection = SFSUtilities.wrapConnection(dataSourceFactory.
      createDataSource(p).getConnection())) {
      Statement sql = connection.createStatement();
      // Clean DB
      sql.execute("DROP TABLE IF EXISTS BUILDINGS");
      sql.execute("DROP TABLE IF EXISTS LW_ROADS");
      sql.execute("DROP TABLE IF EXISTS RECEIVERS");
      sql.execute("DROP TABLE IF EXISTS DEM");
      // Import BUILDINGS
      LOGGER.info("Import buildings");
    }
  }
}

12.2. Connect with Java
57     GeoJsonRead.readGeoJson(connection, Main.class.getResource("buildings.
    →geojson").getFile(), "BUILDINGS");
58
59     // Import noise source
60     LOGGER.info("Import noise source");
61     GeoJsonRead.readGeoJson(connection, Main.class.getResource("lw_roads.
    →geojson").getFile(), "lw_roads");
62
63     // Set primary key
64     sql.execute("ALTER TABLE lw_roads ADD CONSTRAINT lw_roads_pk PRIMARY KEY
    →("PK")");
65
66     // Import BUILDINGS
67     LOGGER.info("Import evaluation coordinates");
68     GeoJsonRead.readGeoJson(connection, Main.class.getResource("receivers.
    →geojson").getFile(), "receivers");
69
70     // Set primary key
71     sql.execute("ALTER TABLE receivers ADD CONSTRAINT RECEIVERS_pk PRIMARY
    →KEY ("PK")");
72
73     // Import MNT
74     LOGGER.info("Import digital elevation model");
75     GeoJsonRead.readGeoJson(connection, Main.class.getResource("dem_lorient.
    →geojson").getFile(), "dem");
76
77     // Init NoiseModelling
78     PointNoiseMap pointNoiseMap = new PointNoiseMap("buildings", "lw_roads",
    →"receivers");
79
80     pointNoiseMap.setMaximumPropagationDistance(160.0d);
81     pointNoiseMap.setSoundReflectionOrder(0);
82     pointNoiseMap.setComputeHorizontalDiffraction(true);
83     pointNoiseMap.setComputeVerticalDiffraction(true);
84     // Building height field name
85     pointNoiseMap.setHeightField("HEIGHT");
86     // Point cloud height above sea level POINT(X Y Z)
87     pointNoiseMap.setDemTable("DEM");
88     // Do not propagate for low emission or far away sources.
89     // error in dB
90     pointNoiseMap.setMaximumError(0.1d);
91
92     // Init custom input in order to compute more than just attenuation
93     // LW_ROADS contain Day Evening Night emission spectrum
94     // LDENConfig ldenConfig = new LDENConfig(LDENConfig.INPUT_MODE.INPUT_MODE_
    →LW_DEN);
95
96     ldenConfig.setComputeLDay(true);
97     ldenConfig.setComputeLEvening(true);
98     ldenConfig.setComputeLNight(true);
99     ldenConfig.setComputeLDEN(true);

(continues on next page)
LDENPointNoiseMapFactory tableWriter = new LDENPointNoiseMapFactory(connection, ldenConfig);

  tableWriter.setKeepRays(true);

  pointNoiseMap.setPropagationProcessDataFactory(tableWriter);
  pointNoiseMap.setComputeRaysOutFactory(tableWriter);

  RootProgressVisitor progressLogger = new RootProgressVisitor(1, true, 1);
  pointNoiseMap.initialize(connection, new EmptyProgressVisitor());

  // force the creation of a 2x2 cells
  pointNoiseMap.setGridDim(2);

  // Set of already processed receivers
  Set<Long> receivers = new HashSet<>();
  ProgressVisitor progressVisitor = progressLogger.subProcess(pointNoiseMap.getGridDim() * pointNoiseMap.getGridDim());
  LOGGER.info("start");
  long start = System.currentTimeMillis();

  // Iterate over computation areas
  try {
    tableWriter.start();
    for (int i = 0; i < pointNoiseMap.getGridDim(); i++) {
      for (int j = 0; j < pointNoiseMap.getGridDim(); j++) {
        // Run ray propagation
        IComputeRaysOut out = pointNoiseMap.evaluateCell(connection, i, j, progressVisitor, receivers);
        // i, j, progressVisitor, receivers);
      }
    }
    finally {
      tableWriter.stop();
    }
  }
  long computationTime = System.currentTimeMillis() - start;
  logger.info(String.format(Locale.ROOT, "Computed in %d ms, %.2f ms per receiver", computationTime, computationTime / (double)receivers.size()));

  // Export result tables as csv files
  CSVDriverFunction csv = new CSVDriverFunction();
  csv.exportTable(connection, ldenConfig.getDayTable(), new File(ldenConfig.getDayTable() + ".csv"), new EmptyProgressVisitor());
  csv.exportTable(connection, ldenConfig.getEveningTable(), new File(ldenConfig.getEveningTable() + ".csv"), new EmptyProgressVisitor());
  csv.exportTable(connection, ldenConfig.getNightTable(), new File(ldenConfig.getNightTable() + ".csv"), new EmptyProgressVisitor());
  csv.exportTable(connection, ldenConfig.getDenTable(), new File(ldenConfig.getDenTable() + ".csv"), new EmptyProgressVisitor());
  } catch (PSQLException ex) {
    if (ex.getCause() instanceof ConnectException) {
      // Connection issue ignore
      LOGGER.warn("Connection error to local PostGIS, ignored", ex);
    } else {
      throw ex;
    }

  12.2. Connect with Java


```java
} catch (SQLException ex) {
    LOGGER.error(ex.getLocalizedMessage(), ex.getNextException());
    throw ex;
}
```

(continued from previous page)
GET STARTED

1. Use Github
2. The repository is here: https://github.com/Ifsttar/NoiseModelling
3. Two principal folders: noisemodelling-emission & noisemodelling-propagation
4. Enjoy & feel free to contact us!
CHAPTER
FOURTEEN

SUPPORT

If you are having issues, please let us know.
You can also contact us at: contact@noise-planet.org
This plugin is distributed under GPL 3 license and is developed by the DECIDE team from the Lab-STIC (CNRS) and by the Mixt Research Unit in Environmental Acoustics (Ifsttar).
INDICES AND TABLES

- genindex
- modindex
- search