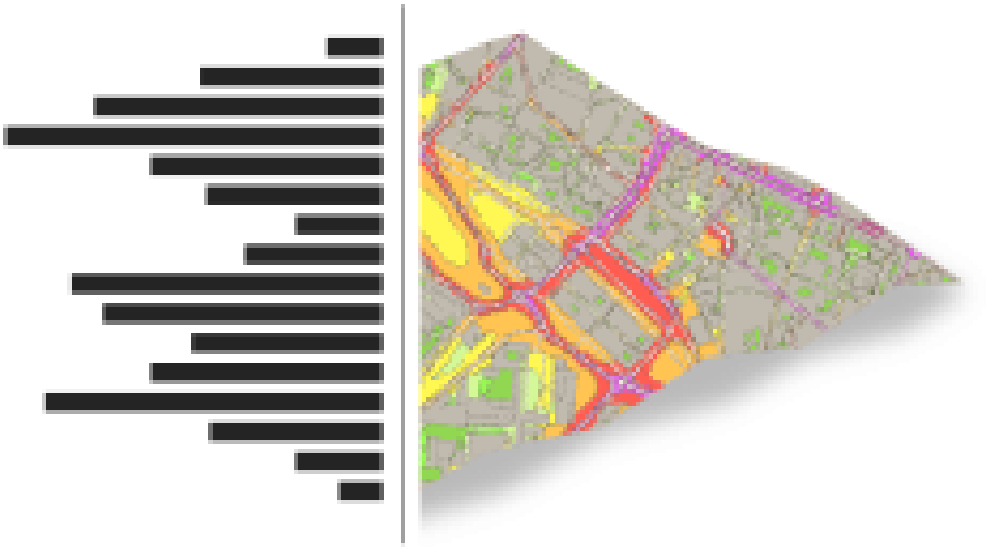
NoiseModelling Documentation

Release 3.3

Aumond P., Fortin N., Le Bescond V.

May 05, 2022

NoiseModelling presentation



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.

A general overview of the model (September 2020) can be found at : https://www.youtube.com/watch?v=V1-niMT9cYE&t=1s

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 offical 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 or a message : https://github.com/Ifsttar/NoiseModelling/discussions

Cite as: *Erwan Bocher, Gwenaël Guillaume, Judicaël Picaut, 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 Veg-DUD (ANR-09-VILL-0007) 2009-2014*
- *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

CHAPTER 1

Numerical Model

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.

CHAPTER 2

Validation

- 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.

CHAPTER 3

Some contributions using NoiseModelling

3.1 Standard Noise maps

- NOURMOHAMMADI, Zahra, LILASATHAPORNKIT, Tanapon, ASHFAQ, Mudabber, et al. Mapping Urban Environmental Performance with Emerging Data Sources: A Case of Urban Greenery and Traffic Noise in Sydney, Australia. Sustainability, 2021, vol. 13, no 2, p. 605. https://www.mdpi.com/2071-1050/13/2/605
- WANG Z., NOVACK T., YAN Y., ZIPF A. Quiet Route Planning for Pedestrians in Traffic Noise Polluted Environments. IEEE Transactions on Intelligent Transportation Systems 2020. https://ieeexplore.ieee.org/document/9139350/keywords#keywords
- BAEZA, Jesús López, SIEVERT, Julia L., LANDWEHR, André, et al. CityScope Platform for Real-Time Analysis and Decision-Support in Urban Design Competitions. International Journal of E-Planning Research (IJEPR), 2021, vol. 10, no 4, p. 1-17. https://www.igi-global.com/article/cityscope-platform-for-real-time-analysis-and-decision-support-in-urban-design-competitions/278826

3.2 Dynamic Noise maps

- CAN A., AUMOND P., BECARIE C., LECLERCQ L. Approche dynamique pour l'étude de l'emprise spatiale du bruit de trafic routier aux heures de pointe, Recherche en Transport Sécurité, 2018.
- CAN A., AUMOND P. BECARIE, C., LECLERCQ, L. Dynamic approach for the study of the spatial impact of road traffic noise at peak hours, Proceedings of the 23rd International Congress on Acoustics, Aachen, Allemagne, 09-13 September, 2019.
- QUNITERO G., AUMOND P., CAN A., BALASTEGUI A., ROMEU J. Statistical requirements for noise mapping based on mobile measurements using bikes. Applied Acoustics, 156, 271-278, 2019.

https://www.youtube.com/watch?v=jl8tASDr-uQ&t=133s

3.3 Probabilistic & Multi-sources Noise maps

- ALIONTE, Cristian-Gabriel et COMEAGA, Daniel-Constantin. Noise assessment of the small-scale wind farm. In : E3S Web of Conferences. EDP Sciences, 2019.
- AUMOND P., CAN A. Probabilistic modeling framework to predict traffic sound distribution, Proceedings of Euronoise, Hersonissos, Crete, 27-31 May 2018
- AUMOND, P., JACQUESSON, L., CAN, A. (2018). Probabilistic modeling framework for multisource sound mapping. Applied Acoustics, 139, 34-43.

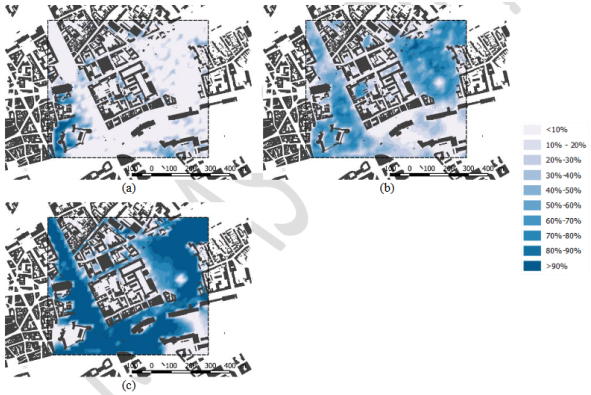
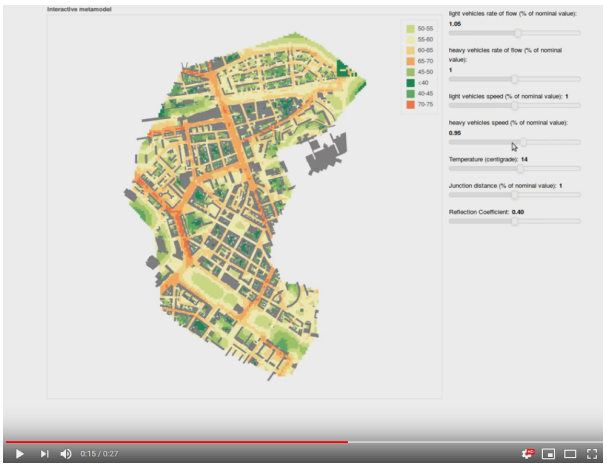


Figure 10 Maps of emerging sound sources for (a) voices; (b) birds; (c) road traffic

3.4 Sensitivity Analysis & data assimilation

- LESIEUR, Antoine, AUMOND, Pierre, MALLET, Vivien, et al. Meta-modeling for urban noise mapping. The Journal of the Acoustical Society of America, 2020, vol. 148, no 6, p. 3671-3681
- AUMOND P., CAN A., MALLET V., GAUVREAU B., GUILLAUME G. Global sensitivity analysis for urban noise modelling Proceedings of the 23rd International Congress on Acoustics, Aachen, Allemagne, 09-13 September, 2019.
- AUMOND P., CAN A., MALLET V., GAUVREAU B., GUILLAUME G. Global sensitivity analysis of a noise mapping model based on open-source software, Proceedings of Internoise 2019, Madrid, Espagne, 16-19 Juin 2019.

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

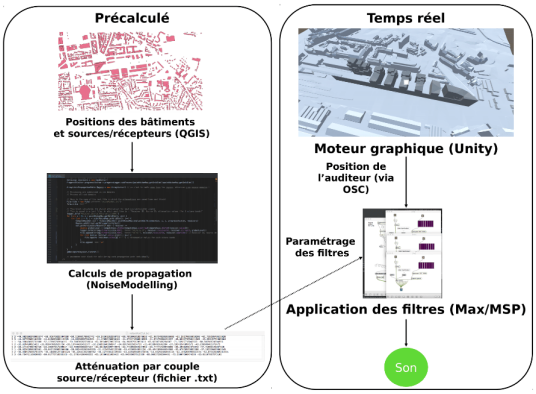


FIGURE 24 – Schéma de principe du prototype de simulateur d'ambiances urbaines. A gauche, les éléments liés à NoiseModelling, qui sont précalculés ; à droite, les éléments temps réel (Unity et Max/MSP).

CHAPTER 4

Get Started - Tutorial

4.1 Requirements:

4.1.1 Windows

Since version 3.3.2, an executable file has been made for you ! You can go directly to step 1.

4.1.2 Other platforms

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 environnement 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_x\`
- If you don't have this result, it is probably because your JAVA_HOME environnement variable is not well settled. To set you JAVA_HOME environnement variable you can adapt (with `x` the JAVA version number) you installed and use the following command line : `setx JAVA_HOME "C:\Program Files (x86)\Java\jre.1.8.x_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 promprt should print `C:\Program Files (x86)\Java\jre1.8.x_x\` whithout the bin directory. If JAVA_HOME is settled as `C:\Program Files (x86)\Java\jre1.8.x_x\bin`, it will not work. It should also point to a JRE (Java Runtime Environment) Java environnement and not JDK.

4.1.3 Docker Setup

When a developer uses Docker (https://www.docker.com/), he creates an application or service, which he then bundles together with the associated dependencies in a container image. An image is a static representation of the application or service, its configuration and dependencies.

A docker image for the NoiseModelling library has already been built. Please visit: https://github.com/tomasanda/docker-noisemodelling

4.2 Step 1: Download the latest release

- Download the latest realease of NoiseModelling on Github.
- *Windows* : you can directly download the executable install file and launch it.
- *Other plaforms* : Unzip the downloaded zip file into a chosen directory.

Warning: 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.

Note:

- Only from version 3.3, NoiseModelling releases include the user interface described in this tutorial.

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.exe for Windows Users
- NoiseModelling\bin\startup.sh for Linux and Mac 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.

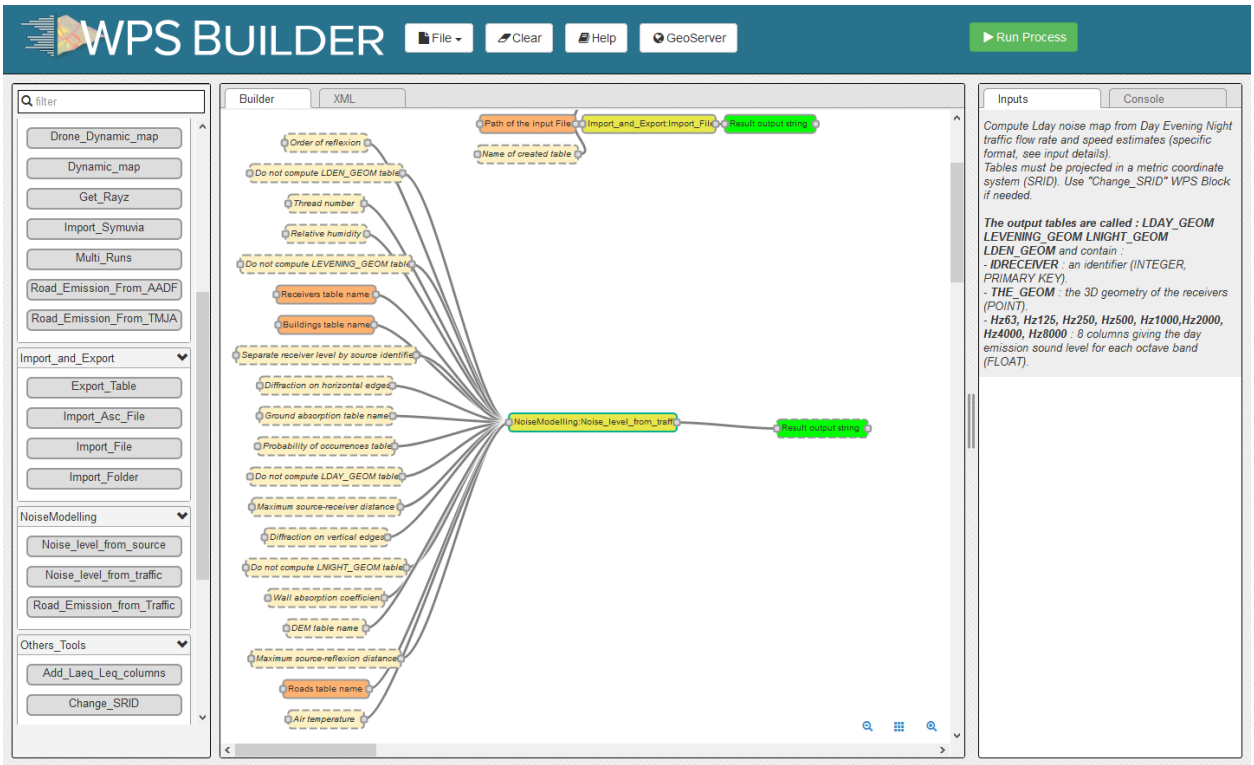
4.6 Step 5: Run Calculation

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.



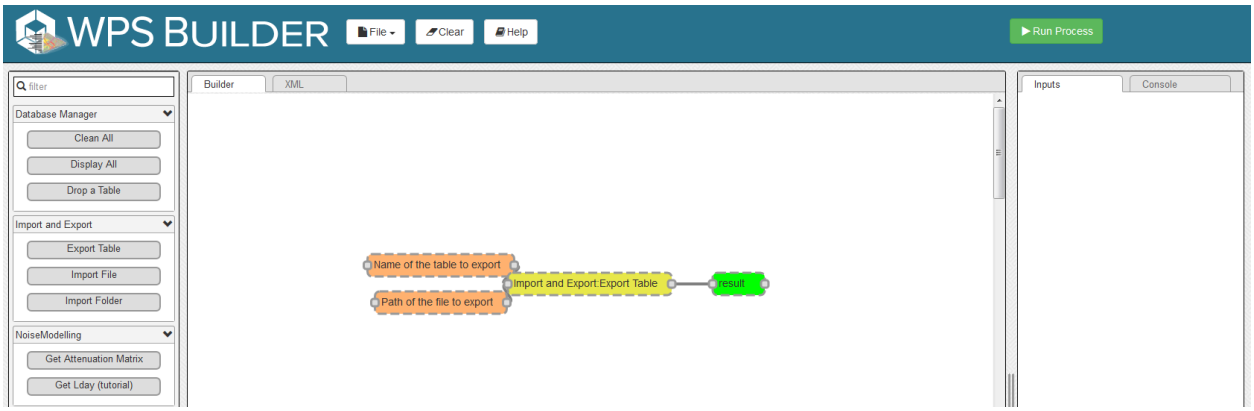
The tables LDAY_GEOM, LEVENING_GEOM, LNIGHT_GEOM and LDEN_GEOM will be created in your database.

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.

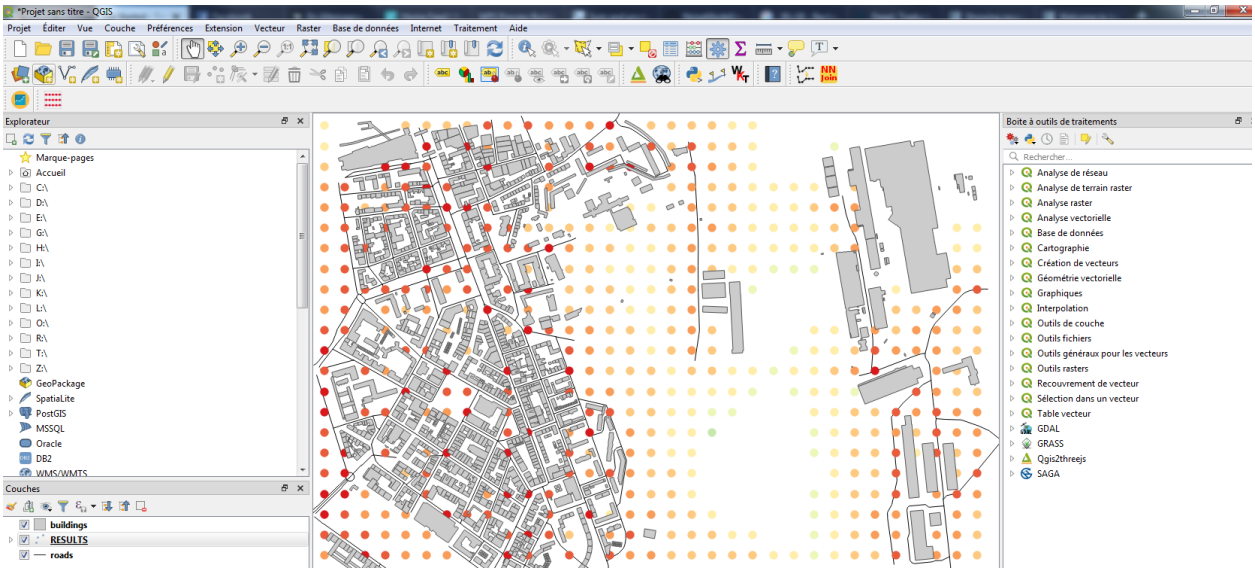
4.7 Step 6: Export (& see) the results

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 syling 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.

CHAPTER 5

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 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.
- As OSM does not include data on road traffic flows, default values are assigned according to the "Good Practice Guide for Strategic Noise Mapping and the Production of Associated Data on Noise Exposure - Version 2" (https://tinyurl.com/1vukv7rj).

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*.

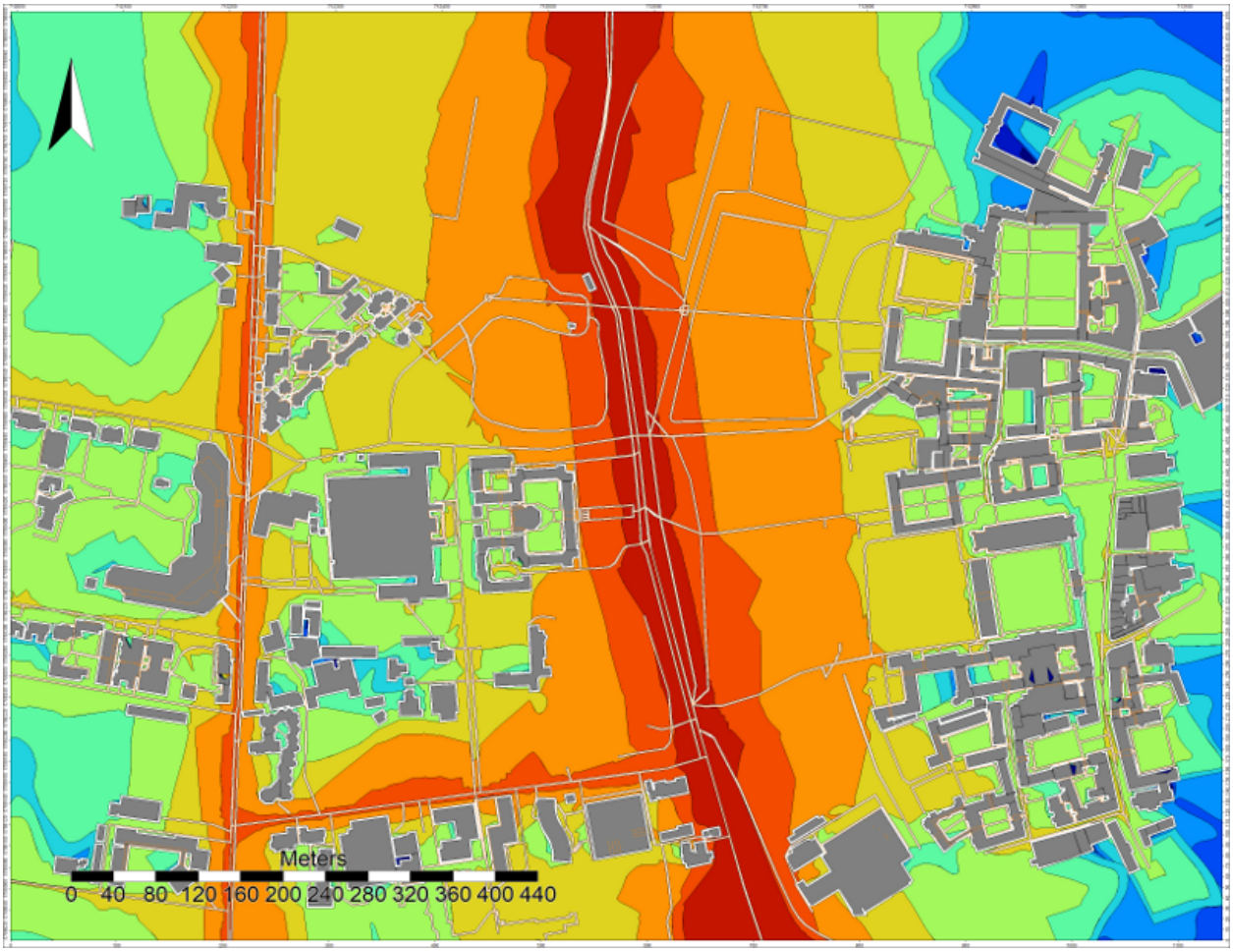
5.7 Step 6: Viewing the result

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.



CHAPTER 6

Matsim - Tutorial

6.1 Introduction

MATSim (https://matsim.org/) is an open-source framework for implementing large-scale agent-based transport simulations. In this tutorial we will learn how to import the output of a successful Matsim simulation into NoiseModelling. The idea is to use the traffic data from Matsim for NoiseModelling road noise emission. Then we will leverage the fact that Matsim is a multi-agent simulator. We will import Matsim agent's positions to calculate their noise exposition throughout the simulated day.

This tutorial we'll look into a simulation of the island in the center of Nantes, the 6th most populated city in France.

6.2 Prerequisites

- You need to have a working installation of the latest NoiseModelling version
- A basic knowledge of what the Matsim traffic simulator does and how it works is preferable
- (optional) A working installation of DBeaver (https://dbeaver.io/) can be useful to visualize the NoiseModelling database tables
- (optional) A working installation of Simunto Via (https://www.simunto.com/via/) can be useful for visualizing the Matsim scenario
- (optional) A working installation of QGis (https://www.qgis.org/) can be useful to visualize resulting GIS data

6.3 The data

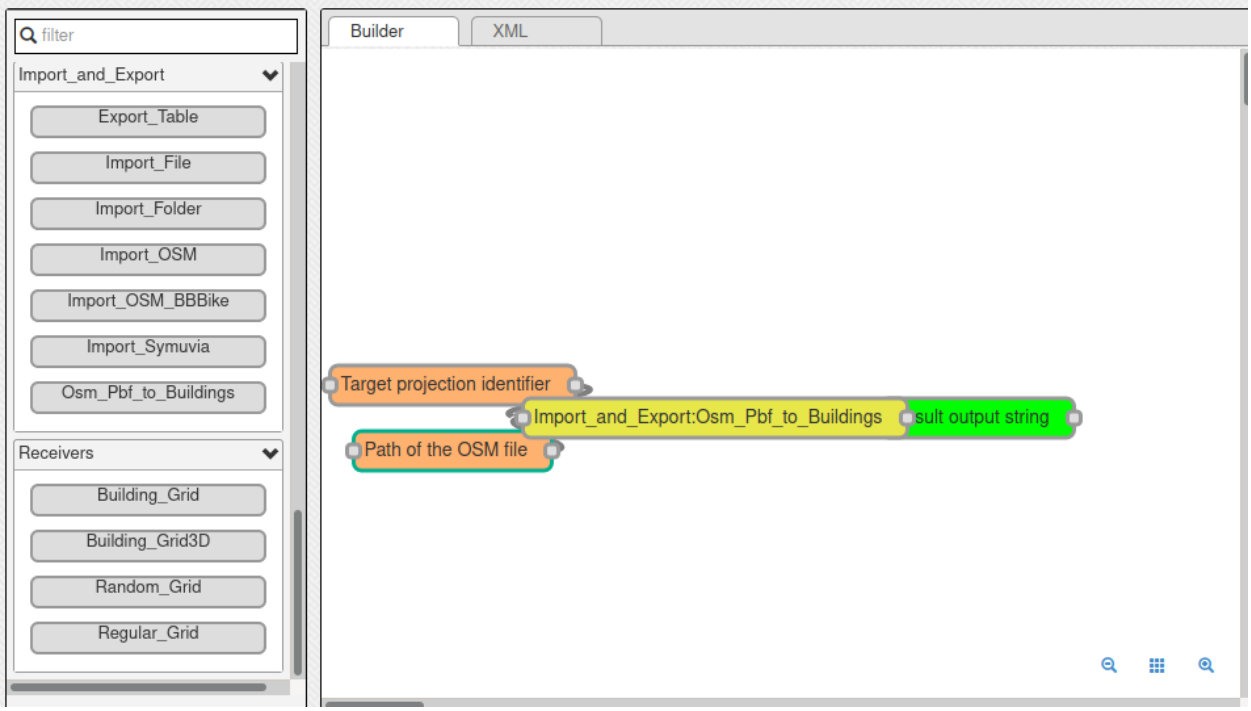
You can download and unzip the data in any folder from here : https://github.com/Ifsttar/NoiseModelling/releases/download/v3.3.1/scenario_matsim.zip

The data folder should contain the following files :

- *nantes_ile.osm.pbf* : the Openstreetmap data of the area. We'll use it to import buildings into NoiseModelling.
- *network.csv* : A file containing the 'true' geometries of the road segments (called "links" in Matsim)
- *output_events.xml.gz* : A file containing the list of Matsim events from the simulation.
- *output_facilities.xml.gz* : A file containing the list of facilities, the agent's activity locations.
- *output_network.xml.gz* : A file containing the Matsim road network, a list of nodes and links.
- *output_plans.xml.gz* : A file containing the list of agents and their final schedule, or plan.

6.4 Step 1 : Import Buildings

The first thing we're going to do is to import buildings. We use the Osm_Pbf_to_Buildings WPS block to do that. Simply put the nantes_ile.osm.pbf path in the 'pathFile' input and set the 'SRID' input to 2154.



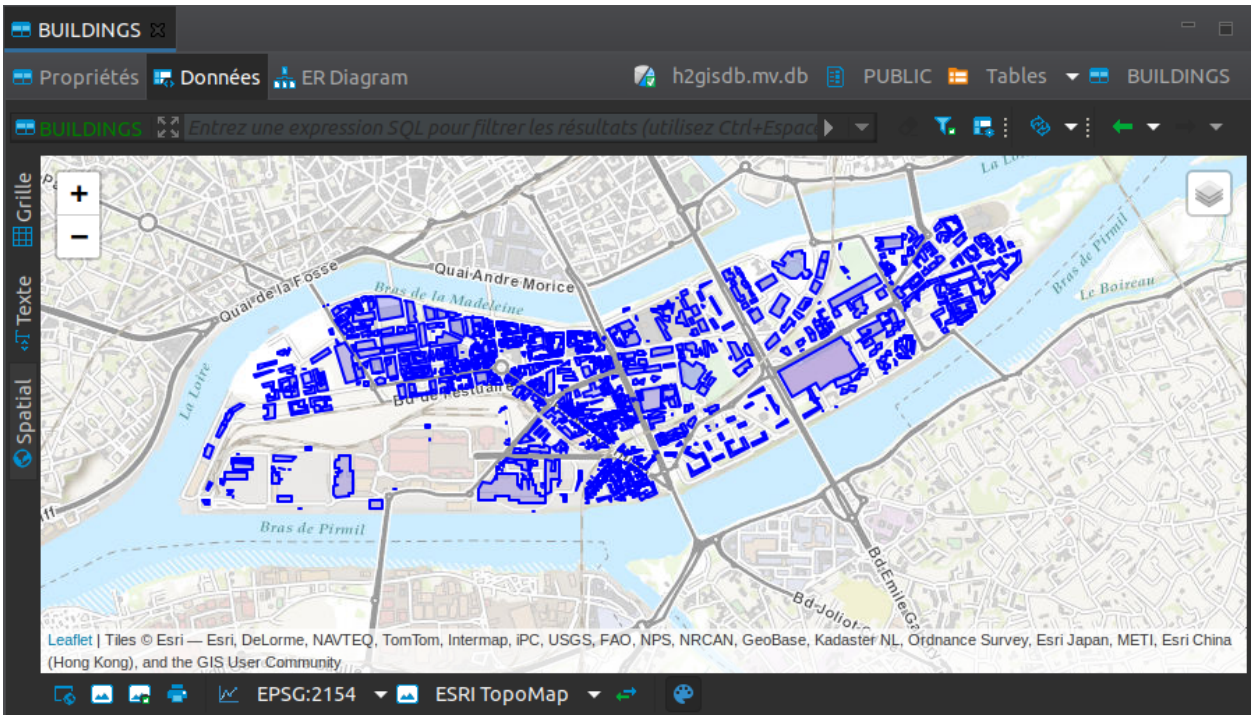
You should end up with a BUILDINGS table containing the island buildings.

6.5 Step 2 : Import Matsim Traffic Data

Now we can import the traffic data from the Matsim simulation. To do that we use the Traffic_From_Events WPS block.

The mandatory inputs are :

- folder : the path of the Matsim folder, here it is where you put the content of the scenario_matsim.zip file
- timeSlice : wich represents the time period you want to aggregate the traffic data over. Here let's use the "quarter" option.



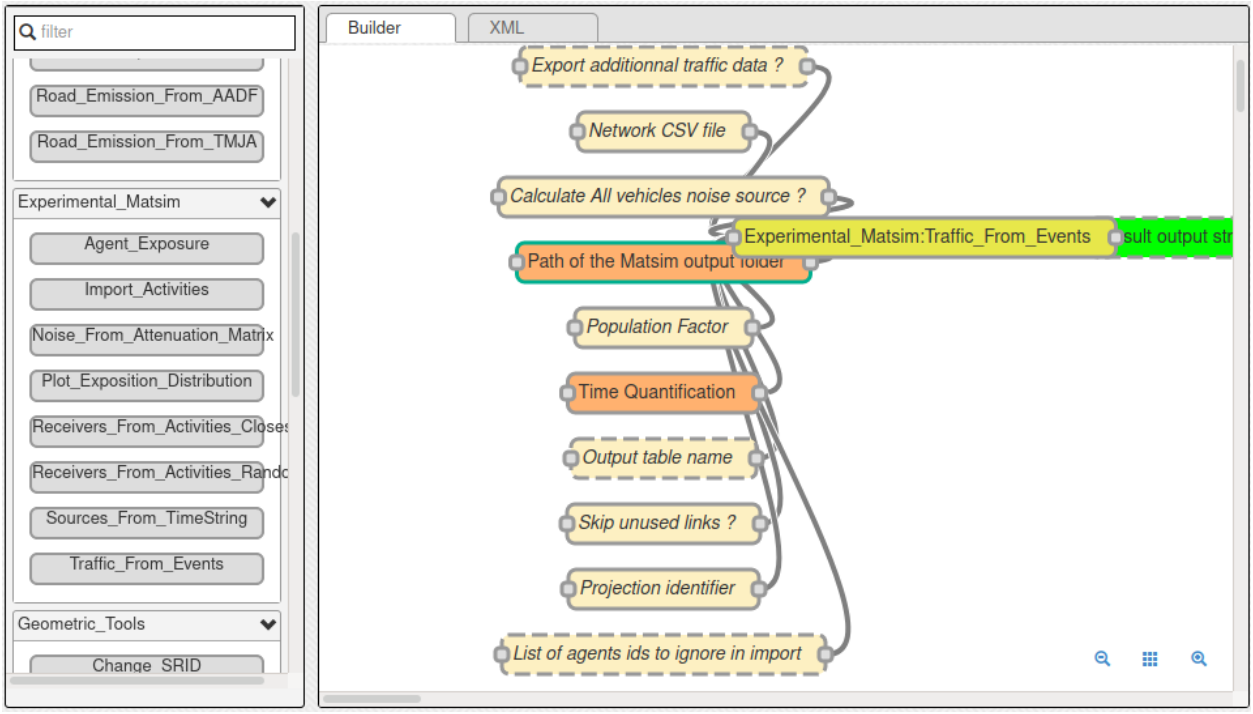
One optional but very important input is the "Network CSV" file path. The idea is that when the matsim scenario was run, the link geometries were simplified to save computation time. This simplification of roads geometry is a bad thing for NoiseModelling since we take buidlings into account (simplified links can pass through buildings) and since source-receiver distance has a big impact on noise levels. That's why the network.csv file is given with the other data files. It contains the "real" geometry of links before Matsim simplification process. (FYI. This is obtained by setting the 'outputDetailedLinkGeometryFile' option to a file name in the pt2matsim config file)

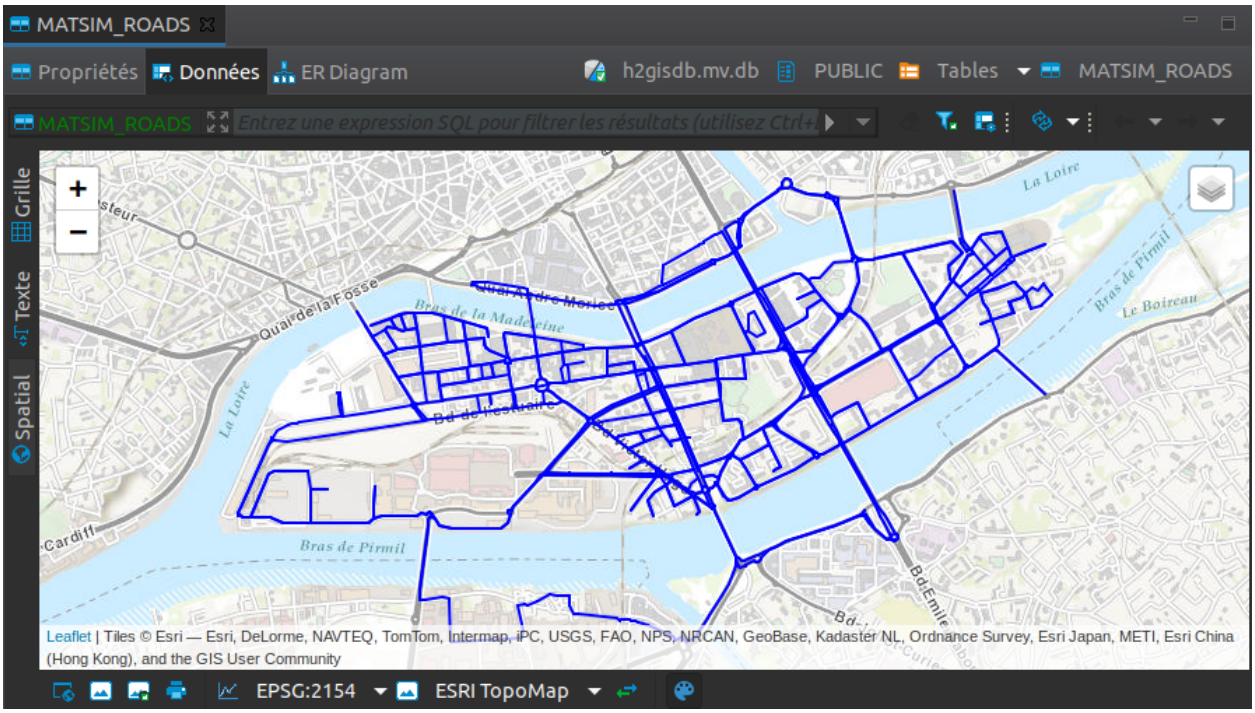
An other important parameter is the 'populationFactor'. This corresponds to the downscaling factor that was used to generate the list of agents. Typically, this list of agents is generated based on the available census and survey data for an administrative area. Here, for our use case, the MAtsim scenario and it's agents were generated by using only 1% of the area total population (that is a population factor of 0.01).

You can explore the other options by reading their descriptions. Here we are going to set them as follows :

- Network CSV file : /path/to/your/scenario_matsim/network.csv
- Export additionnal traffic data ? : false
- Calculate All vehicles noise source ? : true
- Path of the matsim output folder : /path/to/your/scenario_matsim
- populationFactor : 0.01
- Time Quantification : quarter
- outTableName : "" (not set, use default)
- Skip unused links ? : true
- Projection identifier : 2154
- ignoreAgents : "" (not set)

You should end up with a MATSIM_ROADS table containing the links ids and their geometry and a MATSIM_ROADS_STATS table containing the noise power level of each link per 15 min time slice.


filter
Road_Emission_From_AADF
Road_Emission_From_TMJA
Experimental_Matsim
Agent_Exposure
Import_Activities
Noise_From_Attenuation_Matrix
Plot_Exposition_Distribution
Receivers_From_Activities_Close
Receivers_From_Activities_Randc
Sources_From_TimeString
Traffic_From_Events
Geometric_Tools
Change_SRID
Builder
XML
Export additionnal traffic data ?
Network CSV file
Calculate All vehicles noise source ?
Experimental_Matsim:Traffic_From_Events
sult output str
Path of the Matsim output folder
Population Factor
Time Quantification
Output table name
Skip unused links ?
Projection identifier
List of agents ids to ignore in import

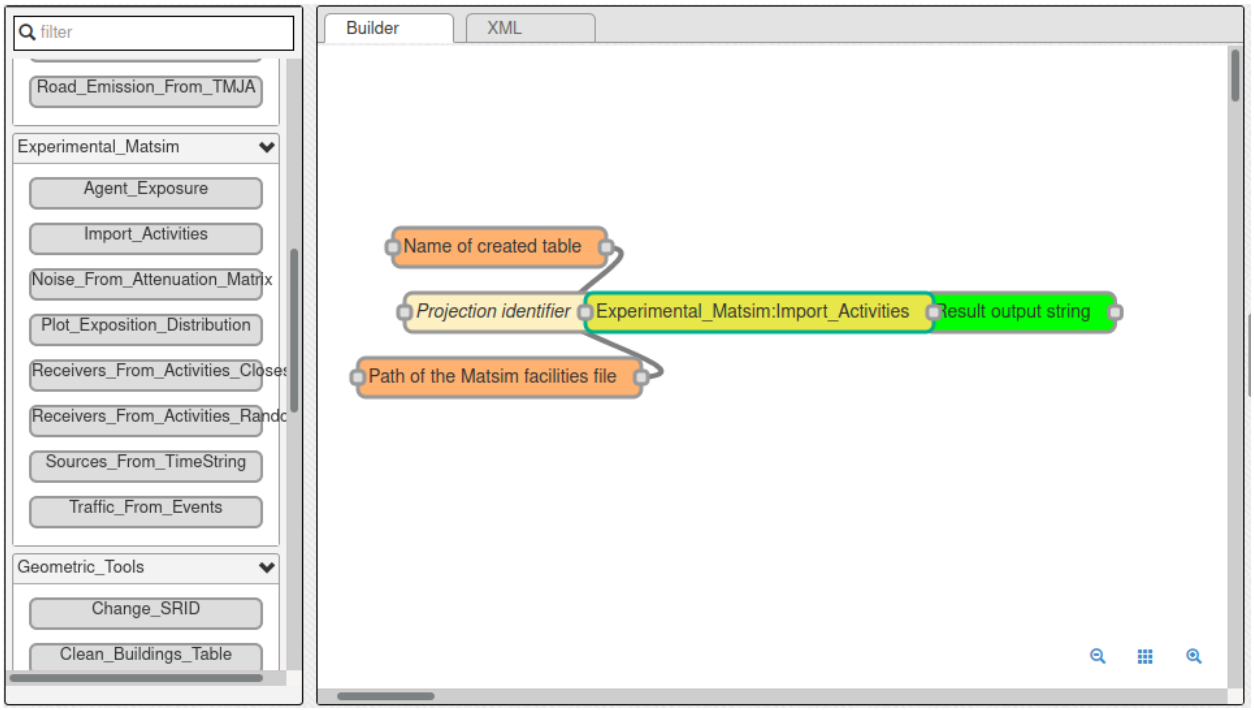

MATSIM_ROADS
Propriétés
Données
ER Diagram
h2gisdb.mv.db
PUBLIC
Tables
MATSIM_ROADS
Entrez une expression SQL pour filtrer les résultats (utilisez Ctrl+E
Grille
Texte
Spatial
La Loire
Bras de la Madeleine
Bras de Pirmil
Le Boireau
Cardiff
Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), and the GIS User Community
EPSG:2154
ESRI TopoMap

6.6 Step 3 : Import Matsim Activities

The next step consists in importing the activities locations from the Matsim simulation.In Matsim, activities are also called facilities.

let's use the Import_Activities WPS bloc. The inputs descriptions are quite straightforward :

- Name of created table : ACTIVITIES
- Projection identifier : 2154
- Path of MatSim facilities file : /path/to/your/scenario_mastim/output_facilities.xml.gz



You should end up with a ACTIVITIES table containing the activities location, and few other properties.

6.7 Step 4 : Assign a Receiver to each Activity

Now, if you look closely, activities are placed in unorthodox locations, sometimes in the river, sometimes in buildings, etc. This is irrelevant for a Matsim simulation but here we want to calculate noise levels, so we need properly placed receivers.

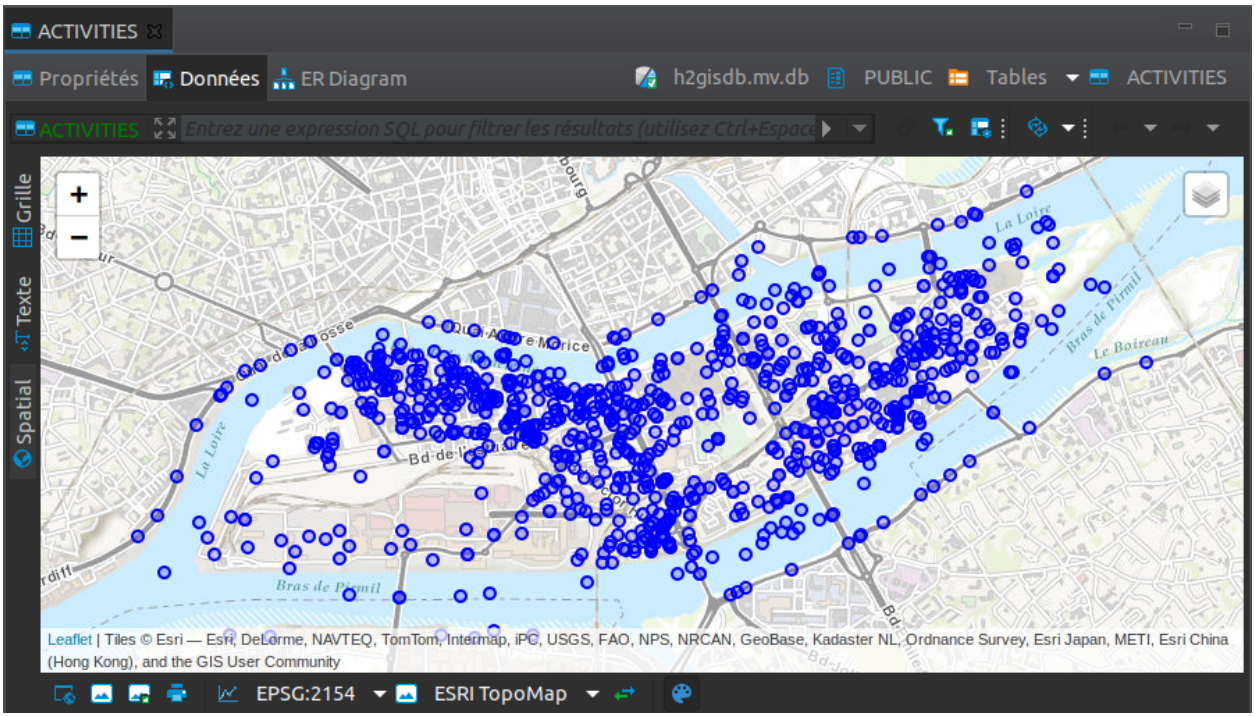
So we want to assign a properly placed receiver for every activity we imported. We do that in 2 steps :

1. we calculate all the "valid" receiver positions using the Building_Grid WPS bloc
2. we choose, for each activity the right receiver.

There are 2 ways to execute step 4.2. We can simply choose the closest receiver for every activity, using the Receivers_From_Activity_Closest WPS bloc. Or we can randomly choose a receiver on the closest building of each activity using the Receivers_From_Activity_Random WPS bloc.

Here we are going to use the latter way, the random one.

Let's calculate all the receivers around our buildings using the Building_Grid WPS bloc with the following inputs :



- Buildings table table : BUILDINGS
- Distance between receivers : 5.0
- height : 4.0

That will place receviers around all the buildings, at 4 meter high and 5 meters apart.

Now, we must use the Receivers_From_Activity_Random WPS bloc. The inputs are simple, you just have to specify the names of the previously created tables

- Name of created table : ACTIVITY_RECEIVERS
- Name of the table containing the activities : ACTIVITIES
- Name of the table containing the buildings : BUILDINGS
- Name of the table containing the receivers: RECEIVERS

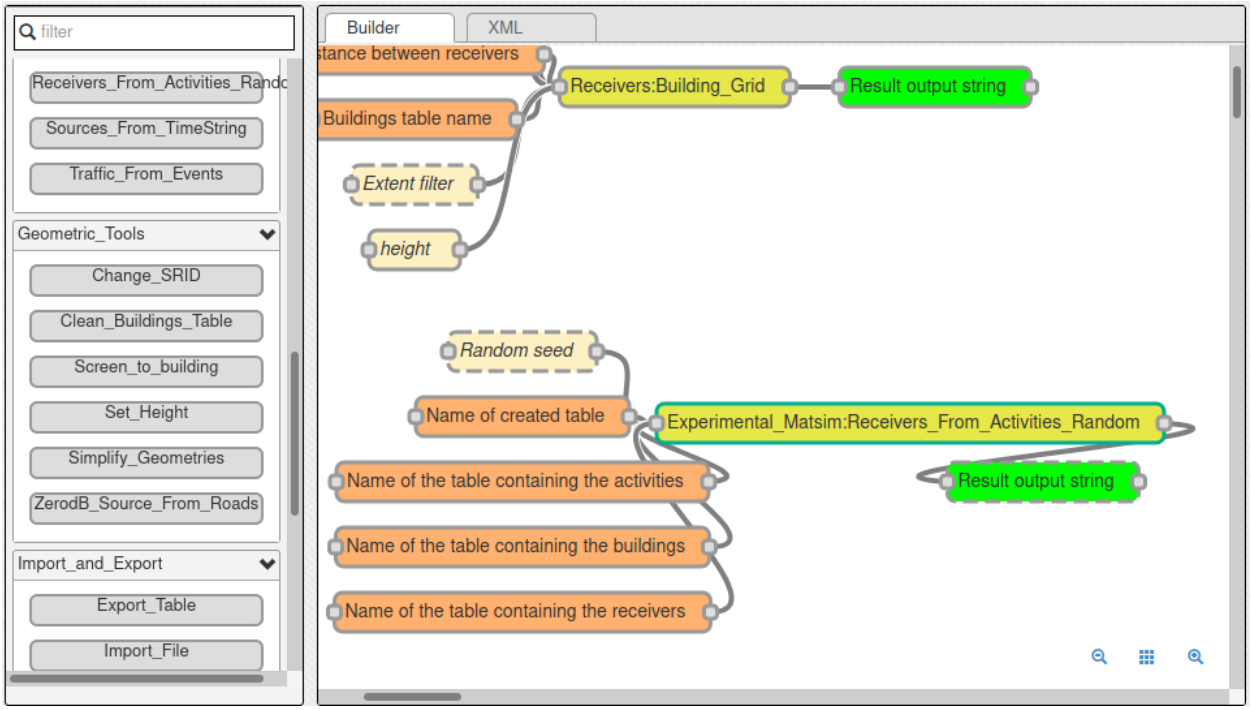
You should end up with a ACTIVITY_RECEIVERS table containing the new location (THE_GEOM, in blue below) as well as the orignal matsim position (ORIGIN_GEOM, in red below). You can inspect the results to see where each activity is placed now.

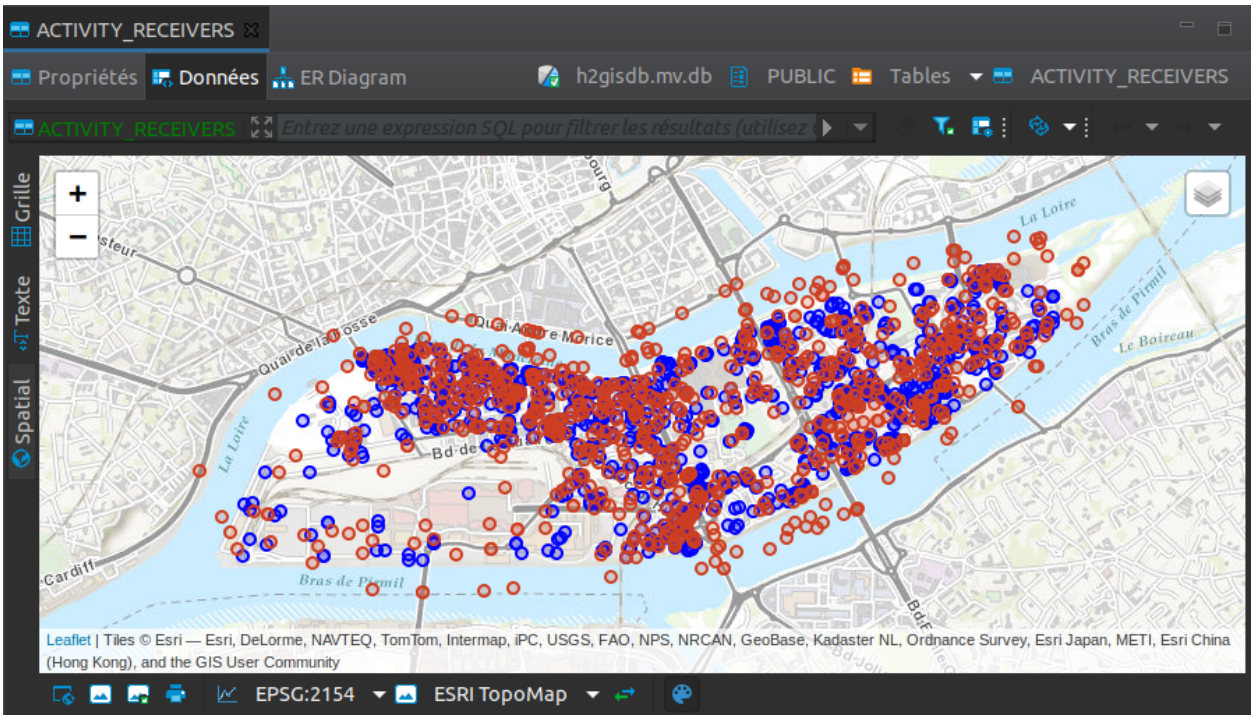
6.8 Step 5 : Calculate Noise Attenuation Matrix

In this step, we want to calculate and store the noise propagation part of NoiseModelling. We need this because we actually have several power spectrum for every road segment, one for every timestep of 15min. In the end we want to have a noise map every 15 minutes (96 maps in total). If we do that directly, by calling something like Noise_level_from_source WPS bloc 96 times, we would be calculating the exact same noise propagation 96 times.

So the process is as follows :

1. we generate a SOURCE table, using the MATSIM_ROADS table, where all levels are set to 0dB.


filter
Receivers_From_Activities_Randc
Sources_From_TimeString
Traffic_From_Events
Geometric_Tools
Change_SRID
Clean_Buildings_Table
Screen_to_building
Set_Height
Simplify_Geometries
ZerodB_Source_From_Roads
Import_and_Export
Export_Table
Import_File
Builder
XML
stance between receivers
Receivers:Building_Grid
Result output string
Buildings table name
Extent filter
height
Random seed
Name of created table
Experimental_Matsim:Receivers_From_Activities_Random
Result output string
Name of the table containing the activities
Name of the table containing the buildings
Name of the table containing the receivers


ACTIVITY_RECEIVERS
Propriétés
Données
ER Diagram
h2gisdb.mv.db
PUBLIC
Tables
ACTIVITY_RECEIVERS
Entrez une expression SQL pour filtrer les résultats (utilisez
Grille
Texte
Spatial
Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), and the GIS User Community
EPSG:2154
ESRI TopoMap

2. We use that table as input of the Noise_level_from_source WPS bloc and setting the 'confExportSourceId' input paramter.

The 'confExportSourceId' parameter will actually ouput, for every recevier, the list of sources that contribute to the resulting levels, with the source-receiver noise attenuation.

We'll then use this attenuation matrix in the next steps to get the 96 noise maps.

6.8.1 Create the 0dB Source table

Here we'll use the ZerodB_Source_From_Roads WPS bloc. It's 2 inputs parameters are quite simple and should be set as follows :

- Input table name : MATSIM_ROADS
- Output table name : SOURCES_0DB

6.8.2 Calculate the attenuation matrix

Let's use the previously generated table to launch our propagation calculation.

As explained before, we'll use the Noise_level_from_source WPS bloc with the 'confExportSourceId' parameter enabled. For more details about the different parameters, browse the NoiseModelling general documentation.

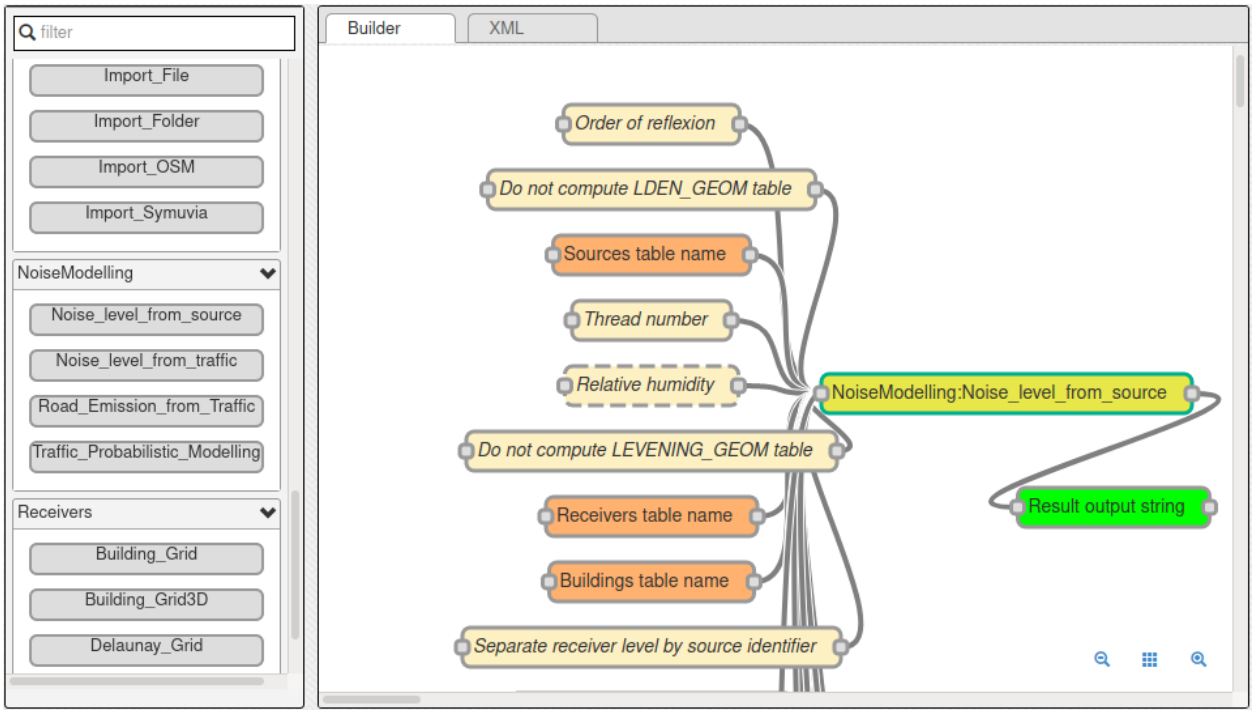
The parameters we will use are the following :

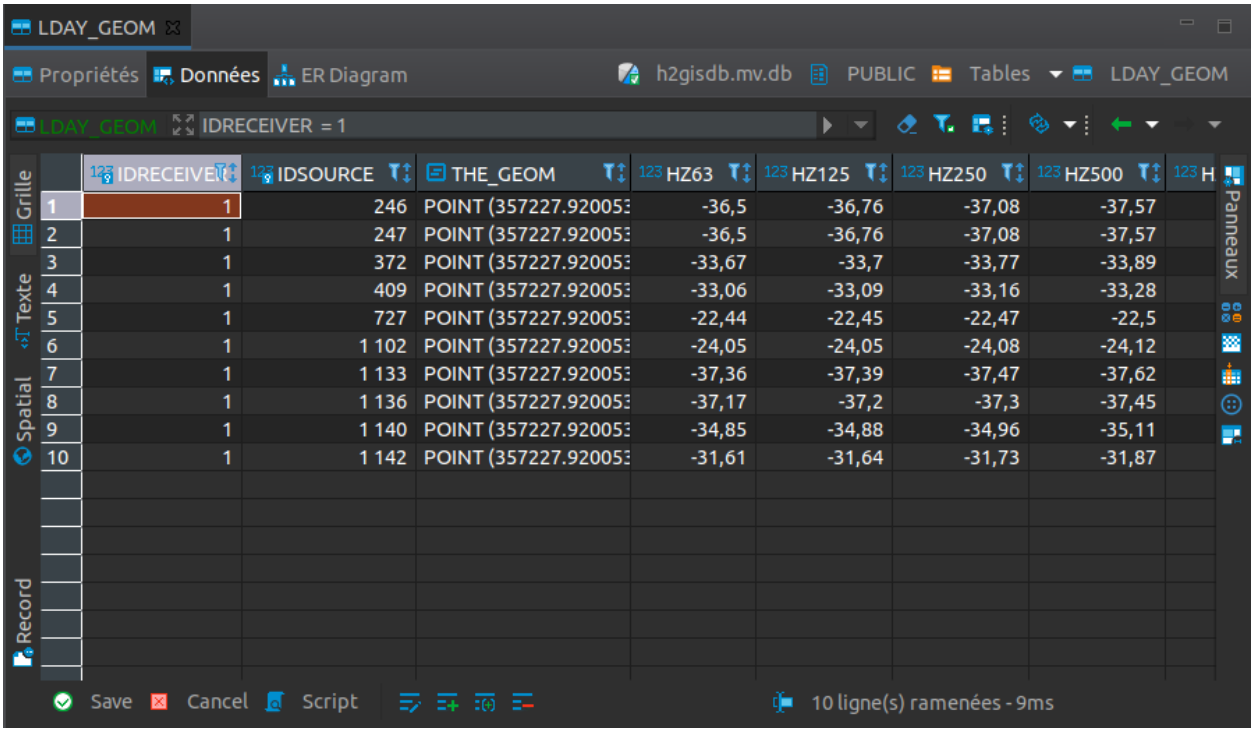
- Buildings table name: BUILDINGS
- Receivers table name : ACTIVITY_RECEIVERS
- Sources table name : SOURCES_0DB
- Maximum source-receiver distance : 250
- Maximum source reflexion distance : 50
- Order of reflexion : 1
- Do not compute LEVENING_GEOM table : true
- Do not compute LNIGHT_GEOM table : true
- Do not compute LDEN_GEOM table : true
- Separate receiver level by source identifier : true
- Diffraction on vertical edges : false
- Diffraction on horizontal edges : true
- Thread number : 4 (your number of available cpu core)

We should end up with a table called LDAY_GEOM that contains a list of contributing source attenuation for every receiver. We can see such a list for the receiver n°1 in the figure below :

6.9 Step 6 : Calculate Noise Maps

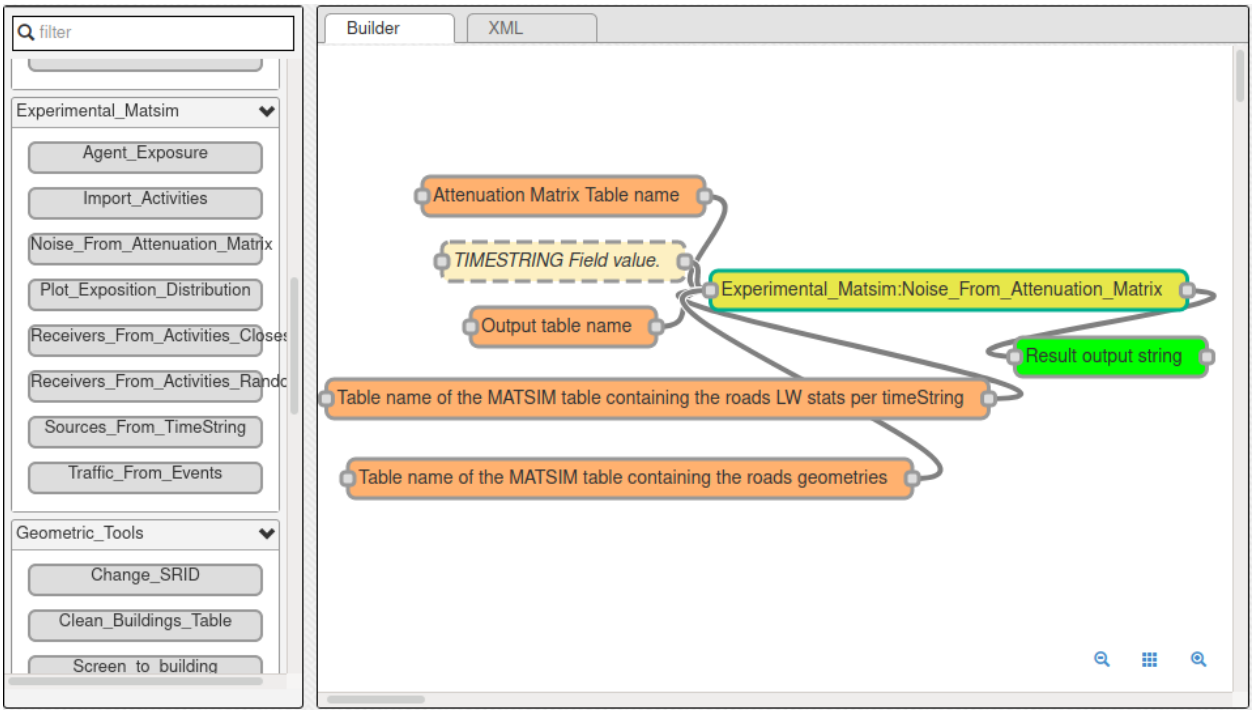
We have noise power levels every 15 minutes in the MATSIM_ROADS_STATS table, and a source-receiver noise attenuation matrix in the LDAY_GEOM table. We just need to combine the two to get receivers noise levels, noise maps, every 15 minutes.


filter
Import_File
Import_Folder
Import_OSM
Import_Symuvia
NoiseModelling
Noise_level_from_source
Noise_level_from_traffic
Road_Emission_from_Traffic
Traffic_Probabilistic_Modelling
Receivers
Building_Grid
Building_Grid3D
Delaunay_Grid
Builder
XML
Order of reflexion
Do not compute LDEN_GEOM table
Sources table name
Thread number
Relative humidity
Do not compute LEVENING_GEOM table
Receivers table name
Buildings table name
Separate receiver level by source identifier
NoiseModelling:Noise_level_from_source
Result output string


LDAY_GEOM
Propriétés
Données
ER Diagram
h2gisdb.mv.db
PUBLIC
Tables
LDAY_GEOM
IDRECEIVER = 1
Grille
Texte
Spatial
Record
Panneaux
IDRECEIVER
IDSOURCE
THE_GEOM
HZ63
HZ125
HZ250
HZ500
1 1 246 POINT (357227.920053 -36,5 -36,76 -37,08 -37,57
2 1 247 POINT (357227.920053 -36,5 -36,76 -37,08 -37,57
3 1 372 POINT (357227.920053 -33,67 -33,7 -33,77 -33,89
4 1 409 POINT (357227.920053 -33,06 -33,09 -33,16 -33,28
5 1 727 POINT (357227.920053 -22,44 -22,45 -22,47 -22,5
6 1 1 102 POINT (357227.920053 -24,05 -24,05 -24,08 -24,12
7 1 1 133 POINT (357227.920053 -37,36 -37,39 -37,47 -37,62
8 1 1 136 POINT (357227.920053 -37,17 -37,2 -37,3 -37,45
9 1 1 140 POINT (357227.920053 -34,85 -34,88 -34,96 -35,11
10 1 1 142 POINT (357227.920053 -31,61 -31,64 -31,73 -31,87
Save
Cancel
Script
10 ligne(s) ramenées - 9ms

This is the purpose of the Noise_From_Attenuation_Matrix WPS bloc. We just have set the right tables as input as follows :

- Attenuation matrix table name : LDAY_GEOM
- Output table name : RESULT_GEOM
- Table name of the MATSIM table containing the roads LW stats per timeString : MATSIM_ROADS_STATS
- Table name of the MATSIM table containing the roads geometries : MATSIM_ROADS



It takes some time but in the end you should get a noise spectrum for every receiver every 15 minutes in the table RESULT_GEOM

We have our noise maps !

6.10 Visualization

6.10.1 Export the data

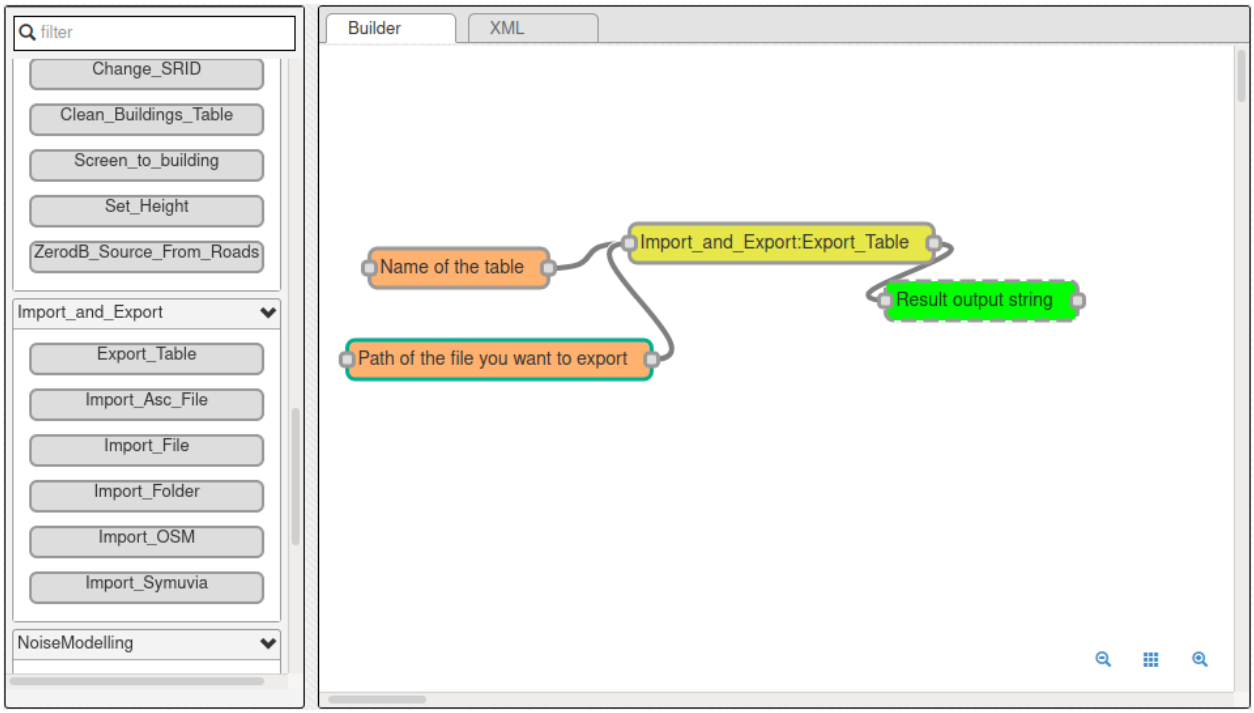
Here we'll look at a nice way to look at the results with QGis.

First we need to export the RESULT_GEOM table data into a Shapefile. We'll simply use the Export_Table WPS bloc with the following parameters :

- Name of the table : RESULT_GEOM
- Path of the file you want to export : /path/to/wherever/results.shp

6.10.2 View it in QGis

Let's go into QGis. We are going to import 2 layers : an osm background and our results.



- In Layer -> Add Layer -> Add vector layer, you can enter the path of your results.shp file. Then click on “add”.
- In Layer -> Add Layer -> Add XYZ Layer, you can add the openstreetmap background.

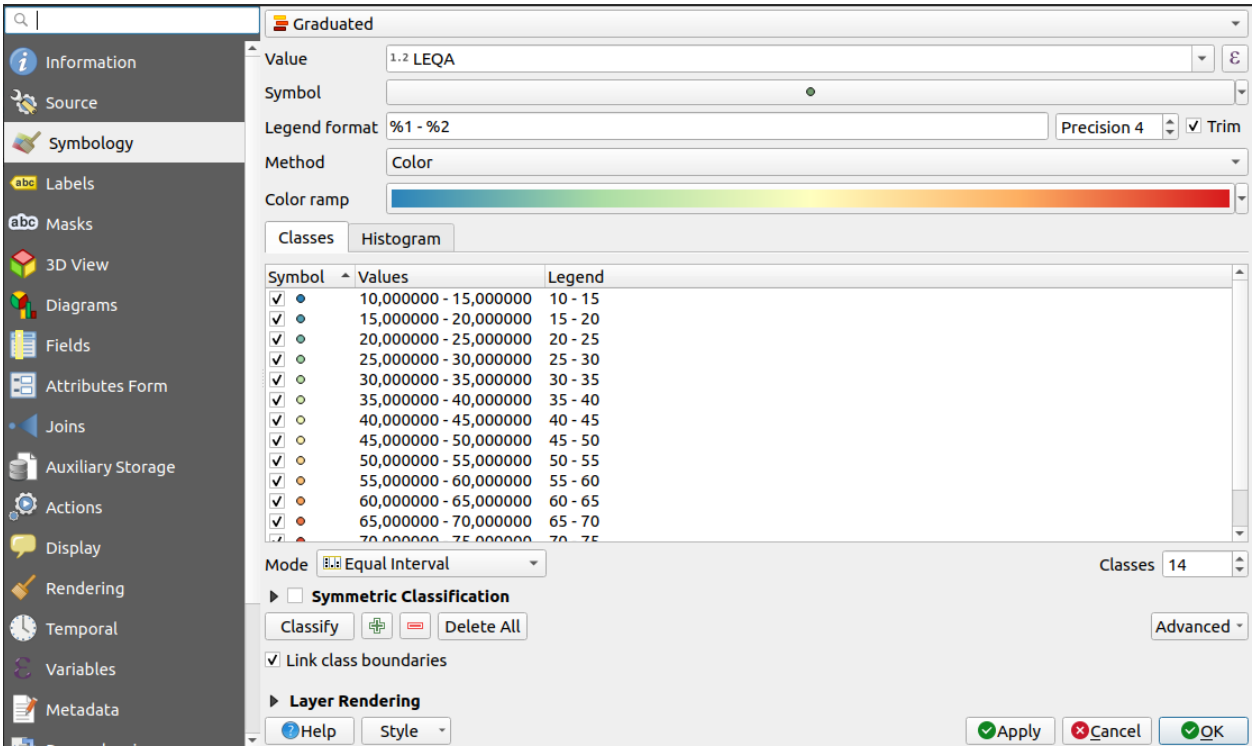
You should see a lot of points all of the same color.

We now need to choose a timeslice we want to visualize, let’s pick 10h00_10h15. If you right click on the receivers layer and click on Filter... you should see the filter dialog.

To filter results for the 10h00_10h15 time period you can enter the following filter query :

The last step is to color the dots based on the LEQA field. Here is my configuration :

And the final result, between 10h and 10h15 :


Graduated
Information
Source
Symbology
Labels
Masks
3D View
Diagrams
Fields
Attributes Form
Joins
Auxiliary Storage
Actions
Display
Rendering
Temporal
Variables
Metadata
Value
LEQA
Symbol
Legend format
%1 - %2
Precision 4
Trim
Method
Color
Color ramp
Classes
Histogram
Symbol
Values
Legend
10,000000 - 15,000000 10 - 15
15,000000 - 20,000000 15 - 20
20,000000 - 25,000000 20 - 25
25,000000 - 30,000000 25 - 30
30,000000 - 35,000000 30 - 35
35,000000 - 40,000000 35 - 40
40,000000 - 45,000000 40 - 45
45,000000 - 50,000000 45 - 50
50,000000 - 55,000000 50 - 55
55,000000 - 60,000000 55 - 60
60,000000 - 65,000000 60 - 65
65,000000 - 70,000000 65 - 70
Mode
Equal Interval
Classes
14
Symmetric Classification
Classify
Delete All
Advanced
Link class boundaries
Layer Rendering
Help
Style
Apply
Cancel
OK

CHAPTER 7

Tutorials - FAQ

7.1 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.

7.2 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.

7.3 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.

7.4 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 recommand using the website https://epsg.io/ to find the appropriate **SRID** code for your location.

CHAPTER 8

WPS Blocks

8.1 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.

8.2 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 !

8.3 Create your own WPS block

Please see Advanced Users Section, because now you want to be one !

CHAPTER 9

WPS Builder

9.1 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

9.2 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

9.3 Can I save my WPSBuilder project ?

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

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

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

9.5 I can't link process block between them ?

It is normal... Still doesn't work !

CHAPTER 10

Create your own WPS block

10.1 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.

10.2 General Structure

10.2.1 1.Import used libraries

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

10.2.2 2.WPS Script meta data

```
title = '....'
description = '.....'
```

10.2.3 3.WPS Script input & output

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

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

10.2.4 4.Set connection method

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

10.2.5 5.Set main method to execute

```
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 ! ']
}
```

CHAPTER 11

Manipulate your database with dBeaver

11.1 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.

11.2 Download dBeaver

You can download dBeaver on the webpage

11.3 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 databse 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 !

11.4 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.

11.5 Connect dBeaver to NoiseModelling libraries

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

CHAPTER 12

Use NoiseModelling by scripting

12.1 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.

12.2 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.

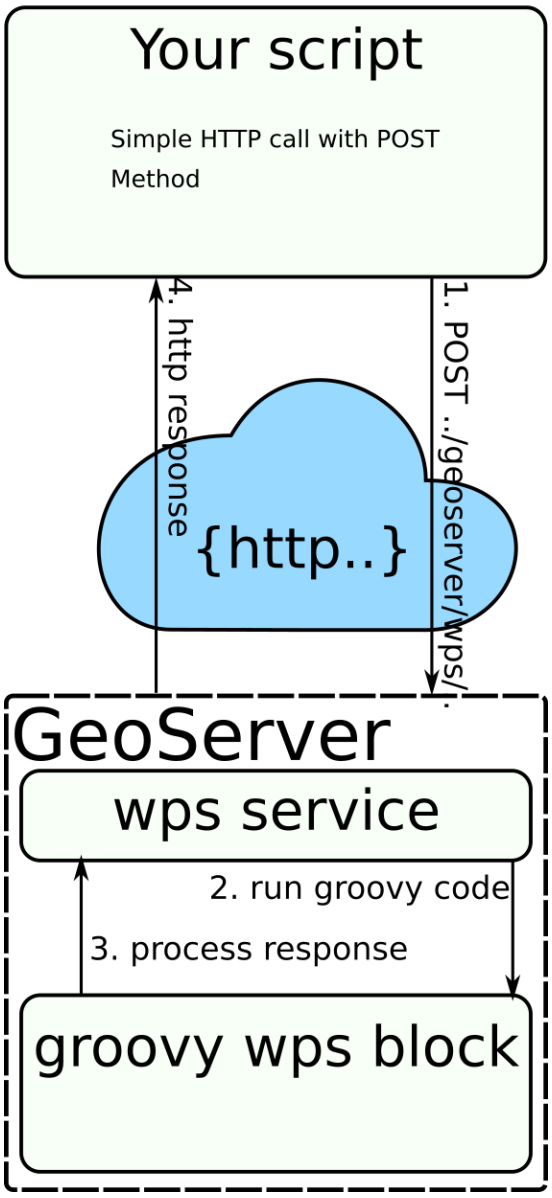
You can first learn more about it:

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

12.3 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.

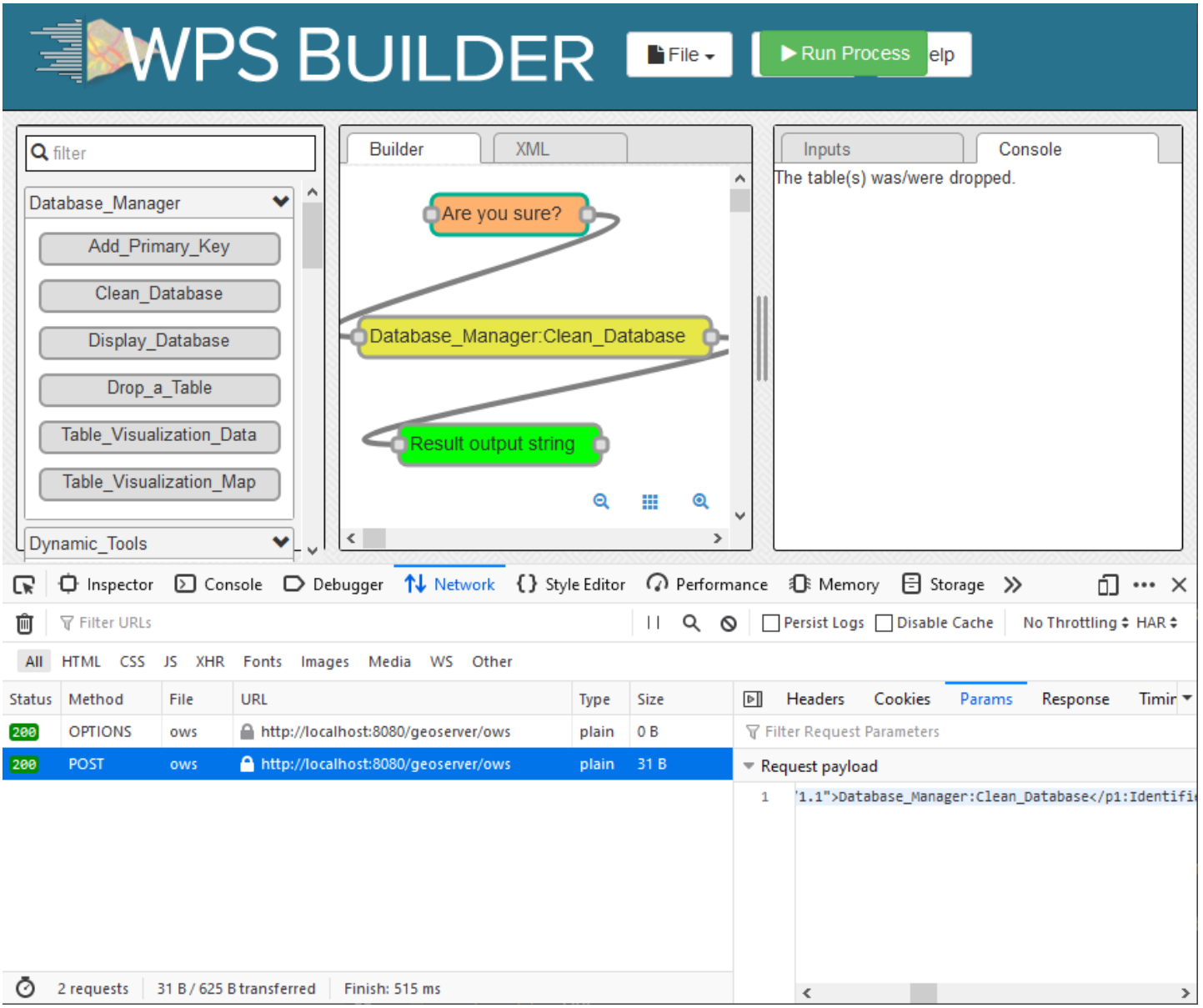

Your script
Simple HTTP call with POST Method
4. http response
1. POST ../geoserver/wps/..
{http..}
GeoServer
wps service
2. run groovy code
3. process response
groovy wps block

12.4 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.

12.5 Get started using Python

The following script run 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.

```
import urllib.request
from string import 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('<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'
                    ':Noise_level_from_traffic</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('<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>$table_to_export</p0:LiteralData></p0:Data></p0:Input><p0:Input><p1:Identifier '
                        'xmlns:p1="http://www.opengis.net/ows/1.1">exportPath</p1:Identifier><p0:Data><p0:LiteralData'
                        '>$export_path</p0:LiteralData></p0:Data></p0:Input></p0:DataInputs><p0:ResponseForm><p0'
```

(continues on next page)

(continued from previous page)

```
                        ':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"}))
```

CHAPTER 13

Use NoiseModelling with a PostGIS database

13.1 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.

13.2 Connect with Java

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

```
<?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>
```

(continues on next page)

(continued from previous page)

```
            <version>1.7.12</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>noisemodelling-emission</artifactId>
            <version>${noisemodelling-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>noisemodelling-propagation</artifactId>
            <version>${noisemodelling-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>h2gis</artifactId>
            <version>${h2gis-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>h2gis-api</artifactId>
            <version>${h2gis-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>h2gis-utilities</artifactId>
            <version>${h2gis-version}</version>
        </dependency>
        <dependency>
            <groupId>org.orbisgis</groupId>
            <artifactId>postgis-jts-osgi</artifactId>
            <version>${h2gis-version}</version>
        </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:

```
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:

```
    public static void main() throws Exception {
        DataSourceFactoryImpl dataSourceFactory = new DataSourceFactoryImpl();
        Properties p = new Properties();
        p.setProperty("serverName", "localhost");
```

(continues on next page)

(continued from previous page)

```
        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();

```

(continues on next page)

(continued from previous page)

```
            // 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");

            GeoJsonRead.readGeoJson(connection, Main.class.getResource("buildings.
↪geojson").getFile(), "BUILDINGS");

            // Import noise source

            LOGGER.info("Import noise source");

            GeoJsonRead.readGeoJson(connection, Main.class.getResource("lw_roads.
↪geojson").getFile(), "lw_roads");
            // Set primary key
            sql.execute("ALTER TABLE lw_roads ADD CONSTRAINT lw_roads_pk PRIMARY KEY␣
↪(\"PK\");");

            // Import BUILDINGS

            LOGGER.info("Import evaluation coordinates");

            GeoJsonRead.readGeoJson(connection, Main.class.getResource("receivers.
↪geojson").getFile(), "receivers");
            // Set primary key
            sql.execute("ALTER TABLE receivers ADD CONSTRAINT RECEIVERS_pk PRIMARY␣
↪KEY (\"PK\");");

            // Import MNT

            LOGGER.info("Import digital elevation model");

            GeoJsonRead.readGeoJson(connection, Main.class.getResource("dem_lorient.
↪geojson").getFile(), "dem");

            // Init NoiseModelling
            PointNoiseMap pointNoiseMap = new PointNoiseMap("buildings", "lw_roads",
↪"receivers");

            pointNoiseMap.setMaximumPropagationDistance(160.0d);
            pointNoiseMap.setSoundReflectionOrder(0);
            pointNoiseMap.setComputeHorizontalDiffraction(true);
            pointNoiseMap.setComputeVerticalDiffraction(true);
            // Building height field name
            pointNoiseMap.setHeightField("HEIGHT");
            // Point cloud height above sea level POINT(X Y Z)
            pointNoiseMap.setDemTable("DEM");
            // Do not propagate for low emission or far away sources.
            // error in dB
            pointNoiseMap.setMaximumError(0.1d);

```

(continues on next page)

(continued from previous page)

```
            // Init custom input in order to compute more than just attenuation
            // LW_ROADS contain Day Evening Night emission spectrum
            LDENConfig ldenConfig = new LDENConfig(LDENConfig.INPUT_MODE.INPUT_MODE_
↪LW_DEN);

            ldenConfig.setComputeLDay(true);
            ldenConfig.setComputeLEvening(true);
            ldenConfig.setComputeLNight(true);
            ldenConfig.setComputeLDEN(true);

            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);
                    }
                }
            } 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.getlDayTable(), new␣
↪File(ldenConfig.getlDayTable()+".csv"), new EmptyProgressVisitor());
            csv.exportTable(connection, ldenConfig.getlEveningTable(), new␣
↪File(ldenConfig.getlEveningTable()+".csv"), new EmptyProgressVisitor());
            csv.exportTable(connection, ldenConfig.getlNightTable(), new␣
↪File(ldenConfig.getlNightTable()+".csv"), new EmptyProgressVisitor());
```

(continues on next page)

(continued from previous page)

```
147             csv.exportTable(connection, ldenConfig.getlDenTable(), new␣
↪File(ldenConfig.getlDenTable()+".csv"), new EmptyProgressVisitor());
148         } catch (PSQLException ex) {
149             if (ex.getCause() instanceof ConnectException) {
150                 // Connection issue ignore
151                 LOGGER.warn("Connection error to local PostGIS, ignored", ex);
152             } else {
153                 throw ex;
154             }
155         } catch (SQLException ex) {
156             LOGGER.error(ex.getLocalizedMessage(), ex.getNextException());
157             throw ex;
158         }
159     }
160 }
```

CHAPTER 14

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 15

Support

If you are having issues, please let us know.

You can also contact us at: contact@noise-planet.org

CHAPTER 16

License

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

CHAPTER 17

Indices and tables

- genindex
- modindex
- search