Starting with UrbanSim: On the Creation of an Introductory Project

Olivier Gallay

Transport and Mobility Laboratory (Transp-OR)
Ecole Polytechnique Fédérale de Lausanne (EPFL)

1 Installation of OPUS (formerly known as UrbanSim) on Windows XP SP2

- **Install the New Opus Installer** (Beta Version) **for UrbanSim**[1] (in May 2010, the latest available version was `4.3.0`), available at:

  `http://www.urbansim.org/Download/WindowsInstaller`

  The file should be named `OPUS-4.3.0.exe` and has a size of around 650 Mb. This version of OPUS/UrbanSim works with **Python 2.6**

  - If asked, choose to **reinstall Python 2.6** (if you have an older version of Python already installed on your computer, it is better to uninstall it)
  - When asked to choose the components:
    - Type of install: **Developper**
    - Install for all users
    - Directories: default
    - Install Python 2.6.2
    - Select: Eclipse, Other, External

[1] In the sequel, we will speak of OPUS and UrbanSim interchangeably.

- OPUS base installation directory:

  `C:\opus`

  (in this folder, you will find after installation a folder named `src` containing the source code and a folder `data` containing the data for the example projects of Eugene and Seattle)

- Eclipse Workspace:

  `C:\workspace`

  (note that Eclipse will also be used to run MATSim)

It is a silent installer, so just let it run (it can take some time)!

- Graphviz Setup Wizard: just confirm and click "next"
- GraphicsMagick Setup Wizard: just confirm and click "next"

N.B: The OPUS Installer should have automatically installed Python additional packages which allow Python to exchange with MySQL (and also with OPUS) on Windows XP SP2. If it is not the case and if you have problems related to that, *c.f.* Section 3.

N.B: It is always possible to run again the Windows Installer to install additional packages or components.

2 Running Examples

- Launch the OPUS GUI (Start/All Programs/OPUS_GUI, or if you want to use the command line: `cd opus; cd src; opus_gui`)

- If you want to incorporate the additional examples given by Paul Waddel in Zurich (in particular `san_antonio_zone`, that will be useful to work thereafter on the Brussels case), you have to:

  - Replace the folder `C:\opus\project_configs` with the new provided `project_configs` folder
  - Replace the folder `C:\opus\src` with the new provided `src` folder
  - Place the provided `san_antonio_zone` and `psrc_parcel` folders in `C:\opus\data`

  OPUS has to be relaunched to be able to access to the additional examples.

- Open the project `san_antonio_zone`
- **Run a simulation**: → "Scenarios" onglet → right click on "san_antonio_baseline" → "Run this scenario" → "Start Simulation"
- **Create an indicator:**
  - → "Results" onglet → right click on "zone_indicator_batch" → Add new indicator visualization:
    * Type: Table
    * Dataset name: zone (for example)
    * Choose and add the indicators you want to draw (with the "+" button)
    * Create (click "Ok")
  - Right click on "zone_indicator_batch" → Run indicator batch on...: selected run or `base_year_data`

3 Data Importation and Exportation From OPUS to MySQL - Setting Up a Working Database

- In order to be able to import and export data to and from UrbanSim, you obviously have to **install a database** on your computer. To install MySQL on Windows XP, you must follow precisely the instructions given at:

  `http://www.urbansim.org/Download/InstallingMySQL`

  During the installation process of MySQL Server, you will have to choose a password for your MySQL platform that we will denote thereafter by `password_1`.

  In brief, throughout the installation process of MySQL, you will have to:

  - Download and run the latest version of the Windows Installer found at:

    `http://dev.mysql.com/downloads`

- Proceed to a “Typical” installation
- You can skip “Sign-Up”
- Then, you have to configure MySQL Server to work with UrbanSim:
  - Choose “Detailed Configuration”
  - Select “Developer Machine”
  - Choose “Non-Transactional Database Only”
  - Select “Manual Setting” with “200” concurrent connections
  - Select “Enable TCP/IP Networking” with Port Number 3306
  - Choose “Standard Character Set”
  - Select “Install As Windows Service” with Service Name “MySQL”
  - Select “Include Bin Directory in Windows PATH”
  - Select “Modify Security Settings” and type as root password:

    `password_1`

At the end of the installation and configuration processes, you still have to **enter precisely the three following commands in the MySQL Command Line Client** (Start/All Programs/MySQL/MySQL Server 5.1/MySQL Command Line Client):

- ```
  GRANT create, delete, drop, index, insert, select,
  update, alter, create temporary tables, file, reload
  ON *.* TO urbansim@localhost IDENTIFIED BY 'password_1';
  ```
- ```
  SET PASSWORD FOR 'urbansim'@'localhost' =
  OLD_PASSWORD('password_1');
  ```
- ```
  FLUSH PRIVILEGES;
  ```

The above three command lines will allow UrbanSim to access the MySQL database. The chosen password for MySQL (*i.e.* `password_1`) has still to be correctly indicated to UrbanSim (it is mandatory since UrbanSim will thereafter access to the MySQL database for data importation and exportation). This can be done either by editing the following `.xml` file:

`C:\opus\settings\database_server_configurations.xml`
  ```

or through the GUI interface where one of the buttons ('Open Database Connection Settings') on the main panel allows us to do so. We have here to modify the following database connection:

mysql_test_database_server

(this database connection will be used thereafter within the OPUS csv_to_sql and sql_data_to_opus tools). For that particular database connection, the host_name has to be set to localhost and the password to password_1 (the user_name remains urbansim, as it should be already fixed). After editing the database_server_configurations.xml file, one must finally have:

```
<mysql_test_database_server setexpanded="True">
<protocol choices="postgres|mssql|mysql|sqlite"
type="string">mysql</protocol>
<host_name type="string">localhost</host_name>
<user_name type="string">urbansim</user_name>
<password type="password">password_1 </password>
</mysql_test_database_server>
```

N.B: Although it has not been useful in my case (hence I have not had to do that), Nicolas Coulombel indicates that after completing the set up of MySQL, it might be necessary to install the executable found at:

http://www.technicalbard.com/files/MySQL-python-1.2.2.win32-py2.6.exe

in order to solve some communication issues between Python and MySQL (see Windows installation notes on the UrbanSim website for more information).

- Before being able to use the data importation and exportation tools of UrbanSim, one has to **create first a database in MySQL**. To do this, write the following command in the MySQL Command Line Client:

  CREATE DATABASE brussels;

  Warning: **no capital letters should be used in the name of the databases, nor in the names and entries of the tables composing the databases** (otherwise errors will appear in the UrbanSim data importation and exportation tools). To check that the database has been created correctly:

```
SHOW DATABASES;
```

- Use the **csv_to_sql tool** (available in the GUI: Data/Tools/tool_library/ data_conversion_tools, you have to right click on `csv_to_sql tool` and select "Execute Tool...").

  Warning: **no blank should appear in the csv file path** (*e.g.* '`Documents and Settings`' should be replaced by '`Documents_and_Settings`'). To avoid this problem, the csv files to be imported in UrbanSim should be placed directly at the root of the hard drive (*e.g.* `C:\`).

  The `database_server_connection` field must be set to:

  ```
  mysql_test_database_server
  ```

  The database name must be set to `brussels` (the one you just created) and the `output_table_name` should be directly chosen as the one wanted to appear thereafter in UrbanSim (it can be for example `households_for_estimation`).

  N.B: Alternatively, the conversion and importation from `.csv` files to MySQL databases can be done via a MySQL GUI (*e.g.* Navicat Lite, see Section 4).

- Use the **sql_data_to_opus tool** (available in the GUI: Data/Tools/Tool Library/opus_data_import_export). This tool directly converts the data from MySQL into the UrbanSim cache format. Note that OPUS has to be relaunched in order that the imported data are visible in the GUI.

  The `database_server_connection` field must be set to:

  ```
  mysql_test_database_server
  ```

  The database name must be set to `brussels` (the one you just created), the `table_name` must be the one chosen before in the `csv_to_sql` tool (for example `households_for_estimation`), the `opus_data_year` must be set for example to `2001` and the `opus_data_directory` must be:

C:\opus\data\san_antonio_zone\base_year_data

- In order to have a good example of a working and consistent dataset, you can create a database from the san_antonio_zone example. In order to proceed, you have to create first a database san_antonio in your MySQL Server, and then you can use (after having loaded the san_antonio_zone example) the **opus_data_to_sql_tool** with the following parameters:
  - Database Server Connection: mysql_test_database_server
  - Database Name: san_antonio
  - Opus Data Directory: C:\opus\data\san_antonio_zone\base_year_data
  - Opus Data Year: 2005
  - Opus Table Name: ALL

4 Managing Data

- In addition to the MySQL Server that you have installed on your computer, it is useful, in order to manage your data in a more comfortable way, to install a tool to visualize and edit your MySQL databases. I would advise not to use the "official" MySQL Workbench GUI (not easy to familiarize with), but to use instead a tool named **Navicat Lite**:

  http://www.navicat.com/download/download.html

- After the installation process of Navicat Lite, you have to create a connection between Navicat Lite and your MySQL Server:

  File → New Connection → MySQL...

  - Connection Name: mysql_connect()
  - Host Name / IP Address: localhost
  - Port: 3306
  - User Name: root
  - Password: password_1
  - click on "save password"

- To open the connection to your MySQL Server from Navicat Lite, you have to right click on mysql_connect() → Open Connection. From that point, all the databases of your MySQL Server are mirrored on Navicat Lite and all the management of your databases can be done from now via Navicat Lite.

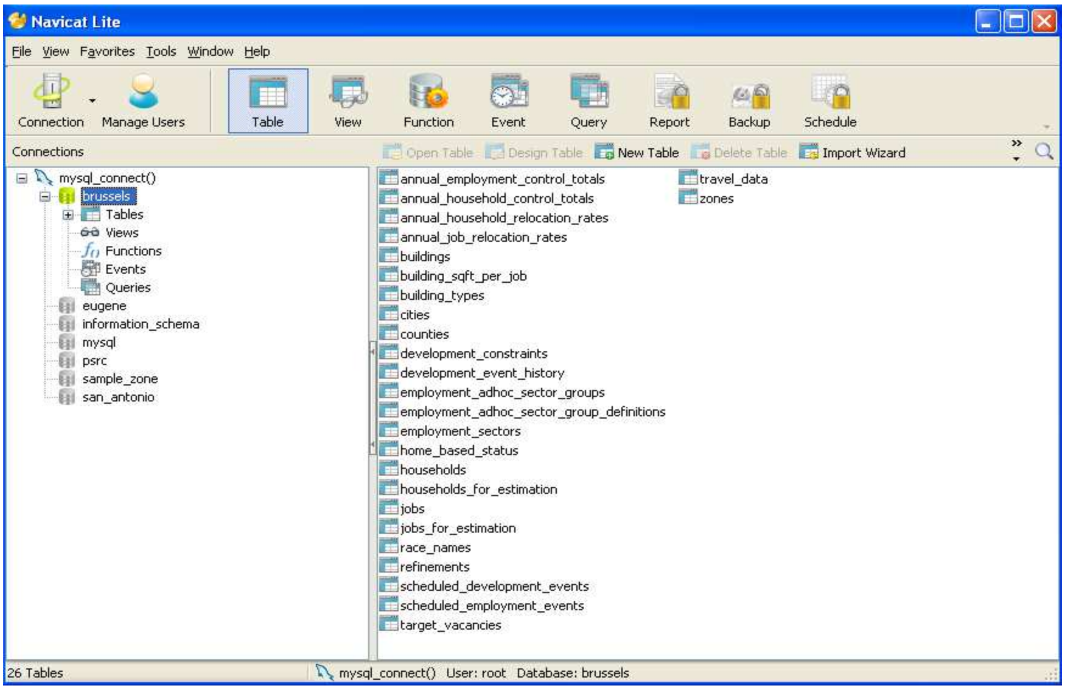


Figure 1: Navicat Lite Exploration Window

- *N.B:* There exists a python script that generates a minimal empty database (basically, this script is generating the mandatory tables, and you have to fill them afterwards) for the zone version of UrbanSim (see Section 6 for more details):

```
create_elixir_schema_for_zone_model.py
```

5 Starting from the San Antonio Example - Some Problems One Might Encounter

- The idea here is to use the `san_antonio_zone` example as a basis and to import the available data for Brussels in order to estimate simple models for Brussels. The available data for Brussels being of course different from the one available for San Antonio, it will take some efforts to compile them in the same format.

Warning: when incorporating the Brussels data into the `san_antonio_zone` example (the data of San Antonio can be removed directly in the folder `C:\opus\data\san_antonio_zone\base_year_data\2005`), the available data for Brussels are for the year 2001 and have thus to be incorporated for the year 2001. A new folder named `2001` must be created in:

```
C:\opus\data\san_antonio_zone\base_year_data\2001
```

and the data available for Brussels will be placed in this folder. Do not forget to change the project configuration file

```
C:\opus\src\san_antonio\configs\san_antonio_zone.xml
```

in the following way:

```
<base_year type="integer">2001</base_year>
```

Alternatively, you might also simply put your available data for Brussels as if they were for year 2005 in the following folder:

```
C:\opus\data\san_antonio_zone\base_year_data\2005
```

- Some data (more precisely some tables) are missing in the dataset available for Brussels in order to be able to estimate and run the different models. Having a look at the set of tables that are mandatory for the zone version of UrbanSim:

  `http://www.urbansim.org/Documentation/Zone/WebHome`

  the situation can be summarized as follows:

  1) **Mandatory tables that exist in the Brussels dataset:**
     * `annual_employment_control_totals`
     * `annual_household_control_totals`
     * `buildings`
     * `development_event_history`

* households
* households_for_estimation
* jobs
* jobs_for_estimation

2) **Mandatory tables that are missing in the Brussels dataset:**

* annual_relocation_rates_for_households
* annual_relocation_rates_for_jobs
* building_sqft_per_job
* building_types
* employment_sectors
* home_based_status
* target_vacancies
* travel_data
* zones

3) **Tables that are not mandatory, but that exist in the Brussels dataset:**

* 2001_sect
* gridcells
* job_tot
* realestate

- Let us now try to estimate a very simple model and to do so let us restrict our attention on a single variable of a particular model (let us consider here the variable `cost_to_income_ratio` within the `household_location_choice` model). Not surprisingly, we get error messages when estimating the model that tables are missing in order to estimate correctly the model. UrbanSim successively indicates that nine tables are actually missing:

  * annual_relocation_rates_for_households
  * building_types
  * target_vacancies
  * zones

  and

  * development_constraint
  * development_types

    * `development_type_group_definitions`
    * `job_building_type`
    * `urbansim_constraints`

- These **missing tables have to be created** somehow and the idea here is to use, for each of the above missing tables, the available table for San Antonio and take inspiration of it to construct a coherent table for Brussels. This can be done by first exporting all the available data for San Antonio from OPUS with the `opus_to_sql` tool. To do this, you first have to create a `san_antonio` database in MySQL. Then, after having opened the `san_antonio_zone` example, export the data of San Antonio from OPUS with the `opus_to_sql` tool:

  – Database Server Connection: `mysql_test_database_server`
  – Database Name: `san_antonio`
  – Opus Data Directory: `C:\opus\data\san_antonio_zone\base_year_data`
  – Opus Data Year: `2005`
  – Opus Table Name: `ALL`

  From this point, the data of San Antonio are available and can thus be managed in Navicat Lite. The missing tables for Brussels can now be copied from San Antonio in Navicat Lite to be modified thereafter to cope with the Brussels case.

  N.B: All the data management can be done with Navicat Lite using SQL queries (alternatively, some missing tables can be created with Matlab or Excel, saved as `.csv` files and then be imported to Navicat Lite[2]) and only when the dataset seems to be finalized and coherent, you might import the dataset into OPUS with the `sql_to_opus` tool, thus placing your mandatory and minimal dataset for Brussels, in OPUS cache format, in:

  `C:\opus\data\san_antonio_zone\base_year_data\2005`

  In that folder, we have now the available data for Brussels and the nine additional missing tables that have been created by taking inspiration from the `san_antonio_zone` example.

[2] *Warning:* Importation in Navicat Lite has to be done from `.txt` files in UTF-8 format.

- At this point, using in some sense data that have been inspired from the `san_antonio_zone` project, you might still have some *compatibility* problems and get the following error message when estimating a simple model (see Fig. 2):

`tuple index out of range`

```
Cache Directory set to: C:\opus\data\eugene_gridcell/base_year_data
Start simulation run: started on Wed May 12 18:14:21 2010
    random seed = 0
    Starting simulation for year 1980: started on Wed May 12 18:14:21 2010
        Closing log file: C:\opus\data\eugene_gridcell/base_year_data\run_model_
system.log
        Logging to file: C:\opus\data\eugene_gridcell/base_year_data\year_1980_l
og.txt
        Simulate year 1980: started on Wed May 12 18:14:21 2010
            Running Household Relocation Model (from urbansim.models.agent_reloc
ation_model): started on Wed May 12 18:14:25 2010....10.7 sec
        Simulate year 1980: completed...................................14.1 sec

        Closing log file: C:\opus\data\eugene_gridcell/base_year_data\year_1980_
log.txt
    Starting simulation for year 1980: completed........................14.5 sec

Start simulation run: completed.........................................14.5 sec

Closing log file: C:\opus\data\eugene_gridcell/base_year_data\run_model_system.l
og
ERROR: Traceback (most recent call last):
  File "C:\opus\src\opus_gui\models_manager\run\run_estimation.py", line 133, in
 run
    self.er.estimate()
  File "C:\opus\src\urbansim\estimation\estimator.py", line 67, in estimate
    self.model_system.run(self.config, write_datasets_to_cache_at_end_of_year=Fa
lse)
  File "C:\opus\src\opus_core\model_coordinators\model_system.py", line 135, in
run
    write_datasets_to_cache_at_end_of_year=write_datasets_to_cache_at_end_of_yea
r)
  File "C:\opus\src\opus_core\model_coordinators\model_system.py", line 279, in
_run_year
    self.vardict[outputvar] = self.do_process(locals())
  File "C:\opus\src\opus_core\model_coordinators\model_system.py", line 360, in
do_process
    return eval(ev)
  File "<string>", line 1, in <module>
  File "C:\opus\src\opus_core\model.py", line 49, in logged_run_method
    results = run_method(*req_args, **opt_args)
  File "C:\opus\src\opus_core\configurable.py", line 38, in config_run_method
    results = run_method(*req_args, **opt_args)
  File "C:\opus\src\urbansim\models\agent_relocation_model.py", line 62, in run
    unplaced_agents = where(agent_set.get_attribute(self.location_id_name) <= 0)
[0]
IndexError: tuple index out of range
```

Figure 2: `tuple index out of range` error message.

for example for a variable entitled

`self.location_id_name`

- While tables inspired from the `san_antonio_zone` project like:

    ∗ `annual_relocation_rates_for_households`

    * `building_types`
    * `development_constraint`
    * `development_types`
    * `development_type_group_definitions`
    * `job_building_type`
    * `target_vacancies`
    * `urbansim_constraints`

should not be the cause of the error stated above as they are mostly providing general attributes for the project. Tables like `zones` for example are likely to be problematic since they involve entries that represent an *identity* that can appear in several other tables (*i.e.* some links are made with other tables using this kind of identity entry). For example, the values of the attribute `zone_id` in the `buildings` table of Brussels might be different from the ones of the attribute `zone_id` in the `zones` table, leading to a `tuple index out of range` error. In that regard, one has to carefully check all the identity entries in all the tables (*i.e.* `xxx_id`), and more generally the entries that appear in more than one table, and check that each value of these entries can be found in all the tables having the same entries. Concerning the available data for Brussels, one has for example to generate a new specific `zones` table for Brussels with the same values of the entry `zone_id` as in the provided `gridcell` table. In particular, one has to **pay special attention to the following entries**:

- `zone_id` in:
    * `zones`
    * `buildings`
    * `jobs`
    * `building_sqft_per_job`
    * `development_constraints`
    * `development_event_history`
    * `scheduled_development_events`
    * `scheduled_employment_events`
    * `travel_data` (here `from_zone_id` and `to_zone_id`)
- `sector_id` in:
    * `employment_sectors`
    * `employment_ad_hoc_sector_group_definitions`

    * jobs
    * jobs_for_estimation
    * annual_employment_control_totals
    * annual_job_relocation_rates
    * scheduled_employment_events
- building_id in:
    * buildings
    * households
    * households_for_estimation
    * jobs
    * jobs_for_estimation
    * scheduled_development_events
    * scheduled_employment_events
- building_type_id in:
    * building_types
    * buildings
    * development_constraints
    * development_event_history
    * target_vacancies
- race_id in:
    * race_names
    * annual_household_control_totals
    * households
    * households_for_estimation
- group_id in:
    * employment_ad_hoc_sector_groups
    * employment_ad_hoc_sector_group_definitions
- household_id in:
    * households
    * households_for_estimation
    * employment_ad_hoc_sector_group_definitions
- job_id in:
    * jobs

    * jobs_for_estimation

  - home_based_status in:

    * home_based_status
    * jobs
    * jobs_for_estimation
    * annual_employment_control_totals

- Even with this new (carefully checked) dataset, containing now all the mandatory tables for the zone version of UrbanSim, you might get an error message of the type:

  missing model specifications

  when estimating the models. This error is due to the fact that **some model specifications are missing in your dataset**. To handle this, you can take these specifications data, in OPUS cache file format, in the original base_year_data folder of San Antonio and copy them into the new base_year_data folder that you have created for Brussels. Alternatively, you might also copy in Navicat Lite these model specifications from the san_antonio database (the one exported from the san_antonio_zone example in OPUS) and paste them in your brussels database. You will also be able to manage and modify these model specifications and coefficients data (to cope with the Brussels case) in Navicat Lite. Do not forget that after the addition or the modification of some tables of the brussels database, you have to export it again to OPUS.

  There are 14 tables that contain the mandatory model specifications and coefficients, which have to be added to the brussels_zone dataset[3]:

  - home_based_employment_location_choice_model_coefficients
  - home_based_employment_location_choice_model_specification
  - household_location_choice_model_coefficients
  - household_location_choice_model_specification
  - non_home_based_employment_location_choice_model_coefficients
  - non_home_based_employment_location_choice_model_specification

[3] As explained later in Section 7, these tables are automatically repopulated after the successful estimation of a model in OPUS.

- non_residential_development_location_choice_model_coefficients
- non_residential_development_location_choice_model_specification
- real_estate_price_model_coefficients
- real_estate_price_model_specification
- residential_development_location_choice_model_coefficients
- residential_development_location_choice_model_specification
- urbansim_constants
- scenario_information

- At this point, the dataset for Brussels contains all the necessary data in order to be able to estimate models[4]. However, you might get the following error message (see Fig. 3):

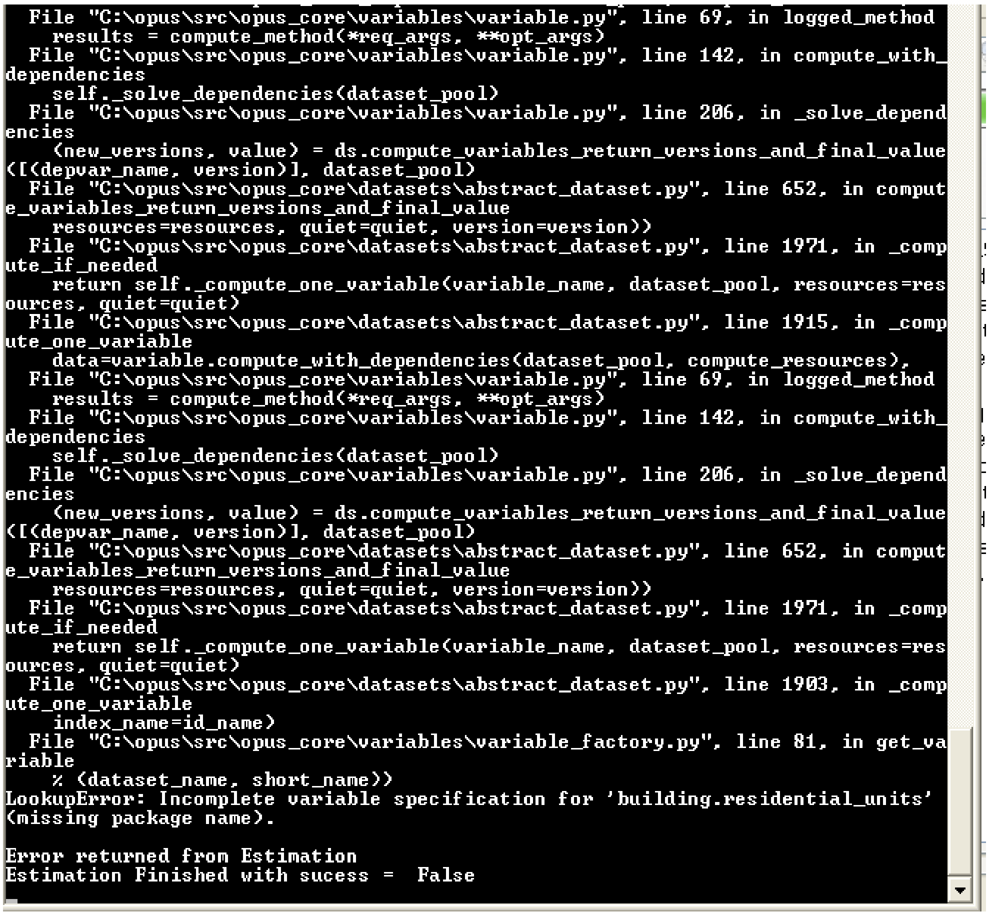
  `missing package name`

  Indeed, **the type and size of the entries in your `brussels` database cannot be chosen freely**. Unless you make the necessary modifications in the corresponding python files, the type and size of the entries composing your tables must be exactly the same compared to the existing entries in the `san_antonio` database. Hence, you have to check carefully, in Navicat Lite, the type and size of all your entries composing your MySQL databases.

  Note that, by default, when you import a `.csv` file into Navicat Lite (*e.g.* when you have created a table in Matlab or Excel), the variable type and size will be set automatically to `text` and `0` for all entries. Hence, you will have to carefully change the type and size for all of these entries to cope exactly with the `san_antonio` database.

  However, even if you fix correctly the type and size of all your entries, some problems might surprisingly occur when exporting your database to OPUS (with the `opus_to_opus` tool). For example, the `buildings` table, which should have entries either of type and size `int,11` or

[4]More precisely, it is actually not mandatory to create the tables `xxx_model_specifications` and `xxx_model_coefficients` in order to be able to estimate the models. Indeed, as explained in Section 7, proceeding to the estimation of the models in OPUS will automatically create these tables. However, it is mandatory to proceed to the estimation of the models (and hence to create these tables) before being able to simulate the models.



```
  File "C:\opus\src\opus_core\variables\variable.py", line 69, in logged_method
    results = compute_method(*req_args, **opt_args)
  File "C:\opus\src\opus_core\variables\variable.py", line 142, in compute_with_
dependencies
    self._solve_dependencies(dataset_pool)
  File "C:\opus\src\opus_core\variables\variable.py", line 206, in _solve_depend
encies
    (new_versions, value) = ds.compute_variables_return_versions_and_final_value
([(depvar_name, version)], dataset_pool)
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 652, in comput
e_variables_return_versions_and_final_value
    resources=resources, quiet=quiet, version=version))
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 1971, in _comp
ute_if_needed
    return self._compute_one_variable(variable_name, dataset_pool, resources=res
ources, quiet=quiet)
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 1915, in _comp
ute_one_variable
    data=variable.compute_with_dependencies(dataset_pool, compute_resources),
  File "C:\opus\src\opus_core\variables\variable.py", line 69, in logged_method
    results = compute_method(*req_args, **opt_args)
  File "C:\opus\src\opus_core\variables\variable.py", line 142, in compute_with_
dependencies
    self._solve_dependencies(dataset_pool)
  File "C:\opus\src\opus_core\variables\variable.py", line 206, in _solve_depend
encies
    (new_versions, value) = ds.compute_variables_return_versions_and_final_value
([(depvar_name, version)], dataset_pool)
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 652, in comput
e_variables_return_versions_and_final_value
    resources=resources, quiet=quiet, version=version))
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 1971, in _comp
ute_if_needed
    return self._compute_one_variable(variable_name, dataset_pool, resources=res
ources, quiet=quiet)
  File "C:\opus\src\opus_core\datasets\abstract_dataset.py", line 1903, in _comp
ute_one_variable
    index_name=id_name)
  File "C:\opus\src\opus_core\variables\variable_factory.py", line 81, in get_va
riable
    % (dataset_name, short_name))
LookupError: Incomplete variable specification for 'building.residential_units'
(missing package name).

Error returned from Estimation
Estimation Finished with sucess =  False
```

Figure 3: `missing package name` error message.

`float,0` (with respect to the `buildings` table of the `san_antonio_zone` example), is impossible to export to OPUS. To solve this problem, the type and size of the entries:

- `non_residential_sqft_capacity`
- `residential_units_capacity`
- `land_area`
- `average_value_per_unit`

have to be fixed to `text` and `0`.

More generally, when an entry (existing in the `san_antonio` database and not in the `brussels` database) is empty in one of your table, the type and size have to be set to `text` and `0`, and not to the type and size of the corresponding entry in the `san_antonio` database. We have to proceed in this way because when an entry is empty, it is automatically set to `WIDEMEMO` or `NULL`, which does not match with an `int` type for example. To avoid such kind of problems, you can also remove in your database all the columns which have only empty values (they are useless and can hence only raise problems).

Warning when creating your dataset (summary):

- One has to be very careful to be consistent with the `xxx_id` entries (*i.e.* `zone_id`, `building_id`, etc.) that appear in several different tables. Indeed, these variables make the link between the different tables (like key attributes do so in MySQL databases).
- The type and size (format) (*e.g.* `float,12` / `text,0` / etc) in the `brussels` database must be exactly consistent with the corresponding entries in the `san_antonio` database. One should double check the newly created database with regard to that in order to avoid thereafter some errors when estimating the models.
- The missing entries in the `brussels` database (in comparison with the `san_antonio` database) should be set to type `text` and size `0` (remember that all the formatting of your database entries can be easily done in Navicat Lite).
- The names chosen for the entries in the tables composing the `brussels` dataset must be consistent (exactly the same) with the names of the existing entries in the corresponding tables composing the `san_antonio` database.

6 Some Useful Facts about OPUS / UrbanSim

- OPUS possesses a Python based modularized architecture that facilitates the insertion of additional plug-ins.
- OPUS has an architecture by layer:
  - `urbansim` (general layer)
  - `urbansim_zone` (zone layer)
  - `san_antonio_zone` (project layer)

In other words, `san_antonio_zone` is the child of `urbansim_zone`, which is itself the child of `urbansim`. Specifications at the child level (*e.g.* zone layer, then project layer) predominate over the parent level.

- Each of the above layer is composed of Python files (that can be found in `C:\opus\src`) and is covered by an additional `.xml` layer that is actually composed of the following files:

  - `urbansim.xml`: model general specifications, database connections, data tools, etc.
  - `urbansim_zone.xml`: model parameters, model arguments, etc.
  - `san_antonio_zone.xml`: project precise model specifications (variable sets considered for each model or submodel to be estimated), definition of the submodel groups, definition of the variables composing the variable library, estimation parameters, scenario informations, indicator definitions, information on your previously executed runs (gives the access to the results data of the previous simulations), etc.

- OPUS data structure:

  folder in OPUS (within the `base_year_data` folder) ↔ **table**
  file in OPUS (opus cache format `.li4` or `.li6`) ↔ **entry**

  Note that the data, in OPUS cache format, can only be read in OPUS (see Fig. 4). All the data edition must be done outside OPUS, in Navicat Lite for example.

- As explained above, the **specific configuration of a project** (*e.g.* here `san_antonio_zone`, a zone project) is defined in:

  `C:\opus\project_configs\san_antonio_zone.xml`

  and **more general specifications are found in**:

  `C:\opus\src\urbansim_zone\configs\urbansim_zone.xml`

- **Models**: they can be **parent** (name written in black in the GUI) **or child** (name written in blue in the GUI). A parent model is generally defined at the `urbansim` layer or at the `urbansim_zone` layer and not at the project layer. A child model is usually defined at the project layer (*e.g.* `san_antonio_zone` layer).

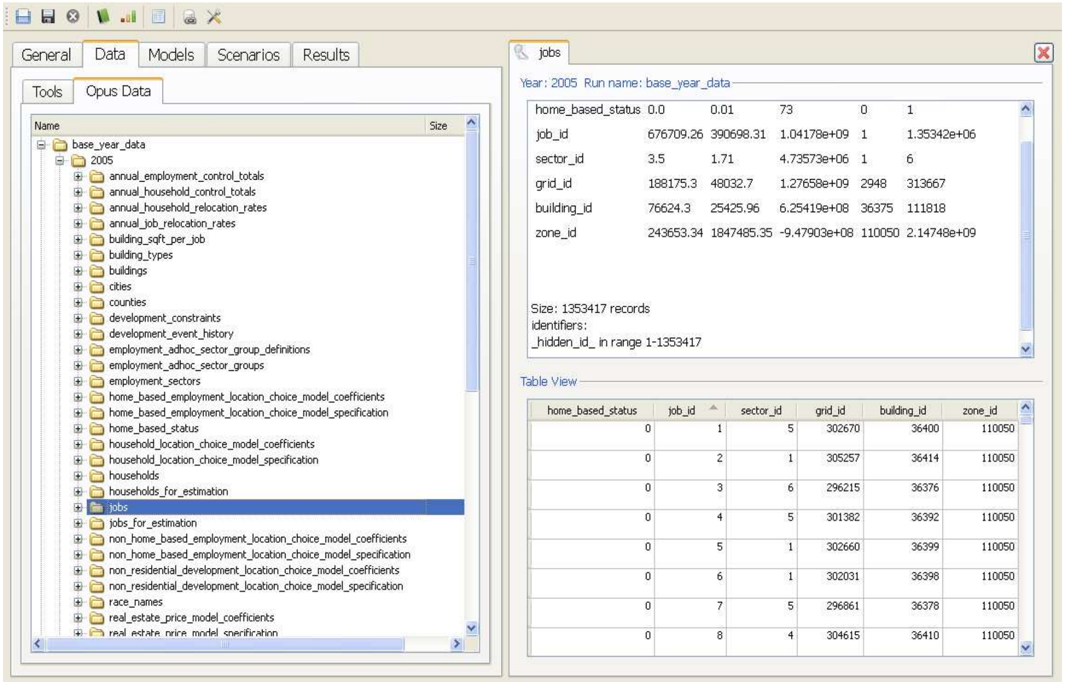


Figure 4: Data visualization in OPUS.

- To **estimate a model**, you have to select the "Models" onglet in the GUI, right click on the model you want to estimate and choose "Run Estimation", and then click on "Start Estimation" on the right part of the window.

- In order to **modify a model**, it has first to be changed to "make node local" (child): this action will create a specific portion of code with the attributes of the parent (*e.g.* from `urbansim_zone.xml` for example) at the project layer (*i.e.* in `san_antonio_zone.xml`) and hence one can modify the model to be specific with the project at hand.

  You can add, modify or remove a variable expression into a model:

  - Expand the model you want to modify
  - Expand "specification"
  - Right click on "submodel" and choose "Edit Submodel"
  - You can add or remove variables
  - You can also create a new variable, *e.g.* the average household size:

`zone.aggregatehousehold.persons, function=mean`

- To **delete a model** (or a submodel), go to the "Models" onglet of the GUI → specification → right click on the model (the submodel) to remove → delete. Of course, one can also remove the specific part of the code dedicated to the model to be deleted in the corresponding `.xml` files. One has here to be careful to remove the specific code in all the `.xml` files (*i.e.* in all layers).
- There exists a difference between model **variables and** model **indicators**: indicators are possible to look at in the result manager.
- **Visualization of a new indicator**:
  - Create your variable using an expression and add it to the variable library in the GUI (see Fig. 6). It is better to check it against data to be sure that the written expression is valid. The variable must be selected to be used as both a Model Variable and an Indicator.
  - In the Results Manager tab of the GUI, you can create a new indicator batch and configure an indicator visualization to display the created variable.
- As explained later, the specifications and variables of the different models are predefined in the different `.xml` files. More precisely, the two files `urbansim.xml` and `urbansim_zone.xml`, the parent files of `san_antonio_zone.xml`, contain general specifications and parameters of the models. The specific configuration of the models will be defined at the project layer, in `san_antonio_zone.xml`.
- When **running a scenario**:
  - The first year and the last year to simulate can be changed in the GUI, or alternatively in `san_antonio_zone.xml`
  - When you expand `san_antonio_baseline` and then "`models_to_run`" → you can choose the models that you want to run during your simulation
- If needed, OPUS tools like the `sql_data_to_opus` tool are also modifiable. To do that, they have to be "make node local".
- There exists **a script that generates a minimal empty database for the zone version of UrbanSim** (all the created mandatory tables are empty and have to be populated thereafter):

```
create_elixir_schema_for_zone_model.py
```

Line 6 in this file has to be put into comment:

```
# metadata.bind =
"sqlite:////Users/pwaddell/sqlite/sample_zone.db"
```

and line 8 has to be uncommented and modified as:

```
metadata.binf="mysql://urbansim:password_1@localhost/sample_zone"
```

A database `sample_zone` has to be created (with the MySQL command line) priorly to the execution of the python script file. To execute the file, open a terminal, go into the folder where the python script is located and execute the following command:

```
python create_elixir_schema_for_zone_model.py
```

This empty database will be useful to determine which tables will be available for Brussels and which table will have to be created. This empty database will also be useful to determine which entries will have to compose the tables and also to check the format (type and size) of these entries.

- The value `-1` for an entry in a table means "ignore that attribute".
- When copying files from another project, do not copy the entire `opus` directory (there might arise some problems thereafter when opening OPUS), but stick to the `src`, `data` and `project_configs` folders.
- It is essential to often make back ups of your project configuration files. Indeed, even an insignificant change to your project (like the creation of an indicator) can make everything crashes...

7 Model Estimation

- One can define the **model specifications** in the "Models" onglet of the GUI (see Fig. 5), or alternatively in the `.xml` files. More precisely, we can choose the set of variables on which the model will be estimated via a linear regression.

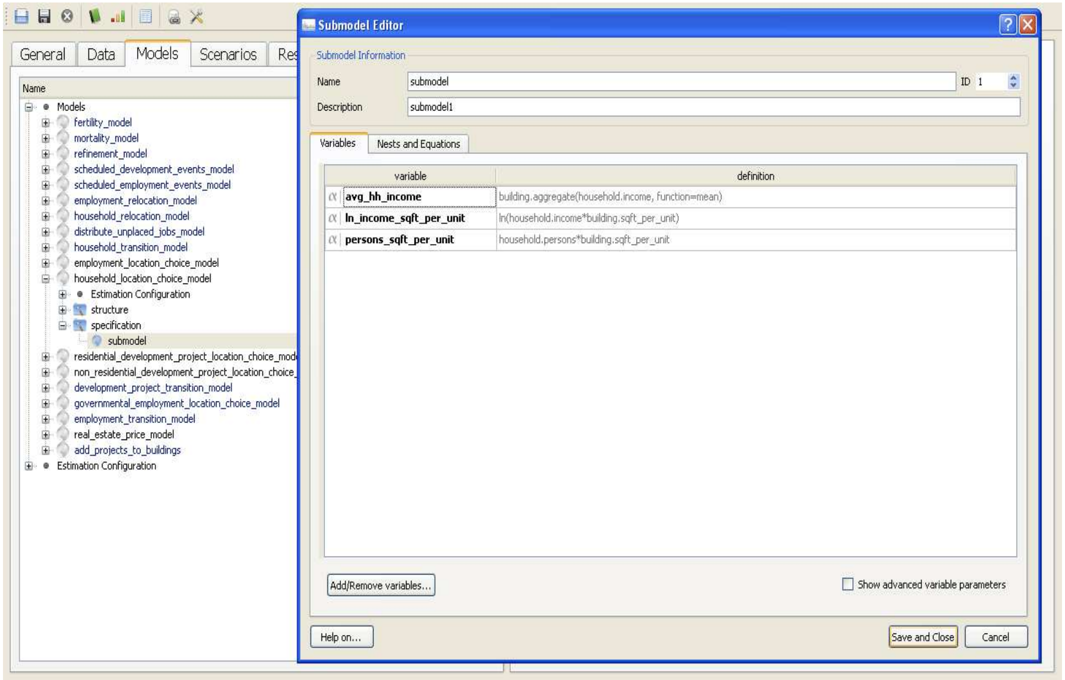


Figure 5: Model definition in OPUS.

- When the variable (or indicator) that one wants to consider is not already contained in the Variable Library, one can **create new** specific **variables or indicators** (see Fig. 6). The list of functions that can be used in the expressions defining variables can be found in the OPUS "Users Guide and Reference Manual" (`www.urbansim.org`).

- Several **model parameters**, including the sample size and the table on which the model will be estimated, can be edited in the "Models" onglet of the GUI (see Fig. 7) or in the file `urbansim_zone.xml`. In particular, it is possible to estimate a model on a different (typically smaller) dataset than the one that will be used for simulation. For example, the estimation dataset used by the `household_location_choice` model is `households_for_estimation`, and not `households`. In order to modify that, one can go on the "Models" onglet of the GUI → `household_location_choice_model` → `structure` → `prepare_for_estimate` → `agents_for_estimation`: `households_for_estimation`.

- As shown in Figs. 8 and 9, when one estimates a model, the estimated coefficients, as well as the specific corresponding model specification, are written (and consequently available) in the `base_year_data`

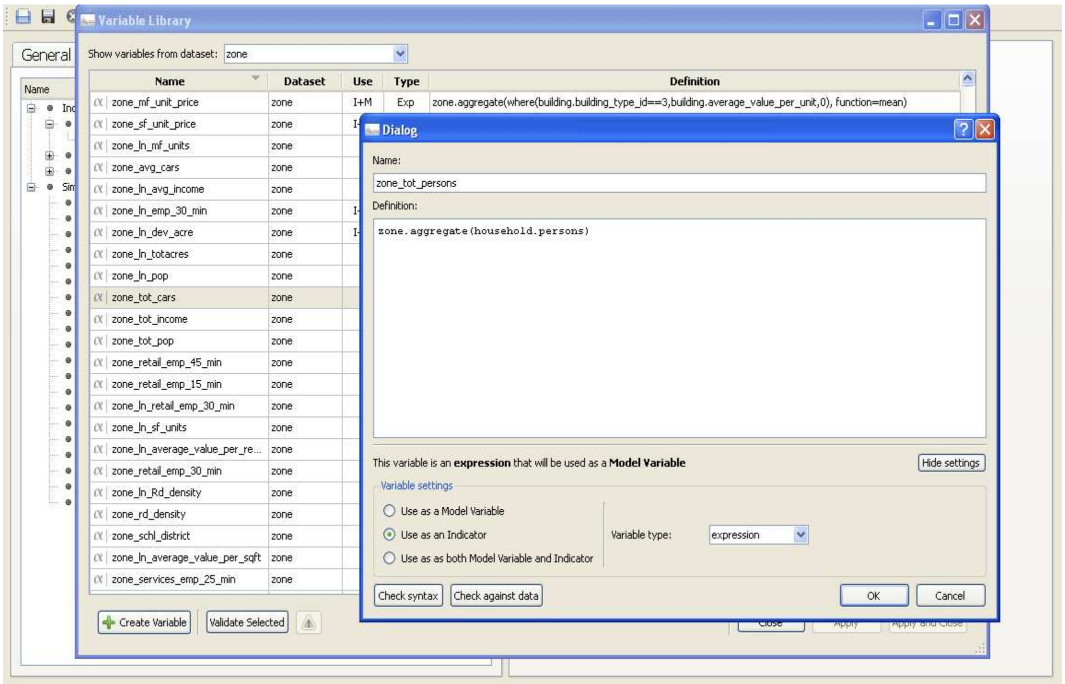


Figure 6: Variable creation in OPUS.

dataset (*e.g.* the `household_location_choice_model_coefficients` and `household_location_choice_model_specification` tables for the `household_location_choice` model). Accordingly, **the estimation of the models could be done outside OPUS** and the estimated coefficients and specification simply written thereafter in the corresponding tables (*e.g.* `household_location_choice_model_coefficients` and `household_location_choice_model_specification`). However, if one wants to proceed like this, one has to be very careful with consistency issues when writing "by hand" in these two particular tables, and especially when writing the specification (*i.e.* the variables) that have to be correctly written in the OPUS expression language (or must be variables that are already existing in the OPUS variable library).

- When one has **specific employment sectors** (*i.e.* `job.sector_id`), like it is the case for Brussels, then the submodels of the `employment_location_choice` model have to be modified accordingly.

- To be able to **estimate a** particular **submodel** (*e.g.* within the `employment_location_choice` model), the dataset used for estimation must contain sufficient number of entries from the type to be estimated

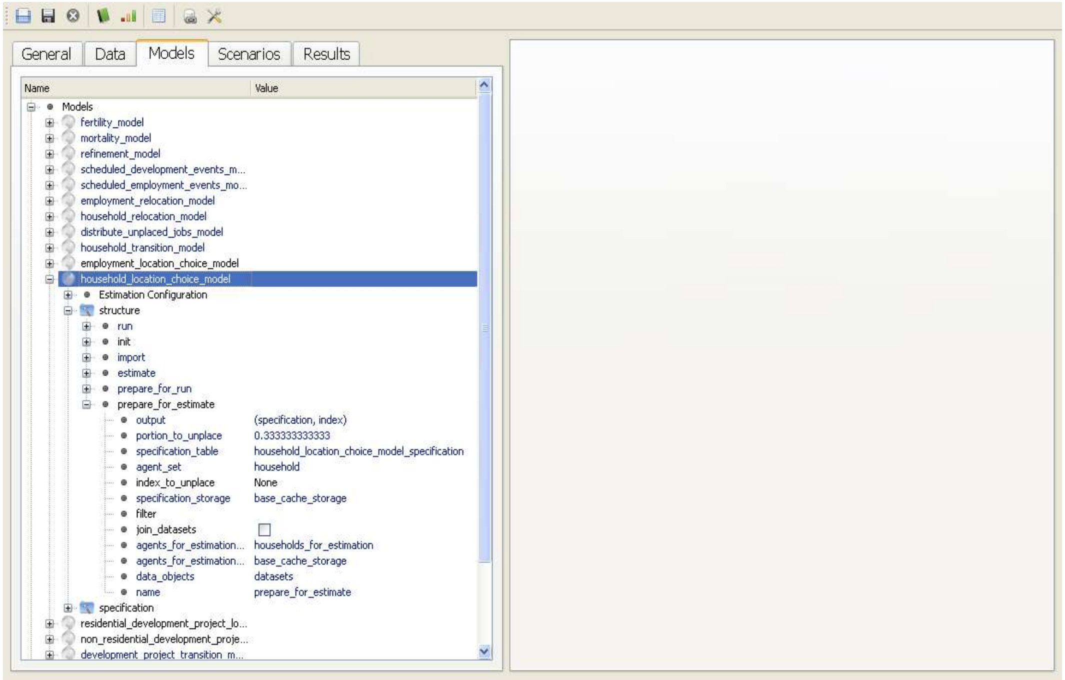


Figure 7: Model parameters in OPUS.

(*i.e.* `sector_id`) in the submodel (here, the `jobs_for_estimation` table is concerned).

- To be able to estimate the `home_based_employment_location_choice` submodel, one must have a sufficient number of home-based jobs (`home_based_status=1`) in your `jobs_for_estimation` table. Otherwise, you get the following error (see Fig. 10):

  `no attribute coefficients error`

- One has to check that the `jobs` table effectively contains a `zone_id` attribute, especially when one wants to proceed to a zone-aggregation of the jobs. OPUS do not seem to be able to link first a job to a building (via the `building_id` attribute) and then a building to a zone (via the `zone_id` attribute).

- With report to the specific building types we have for Brussels, one has to redefine variables like `is_retail` or `is_office`:

  `is_retail = building.building_type_id=4`

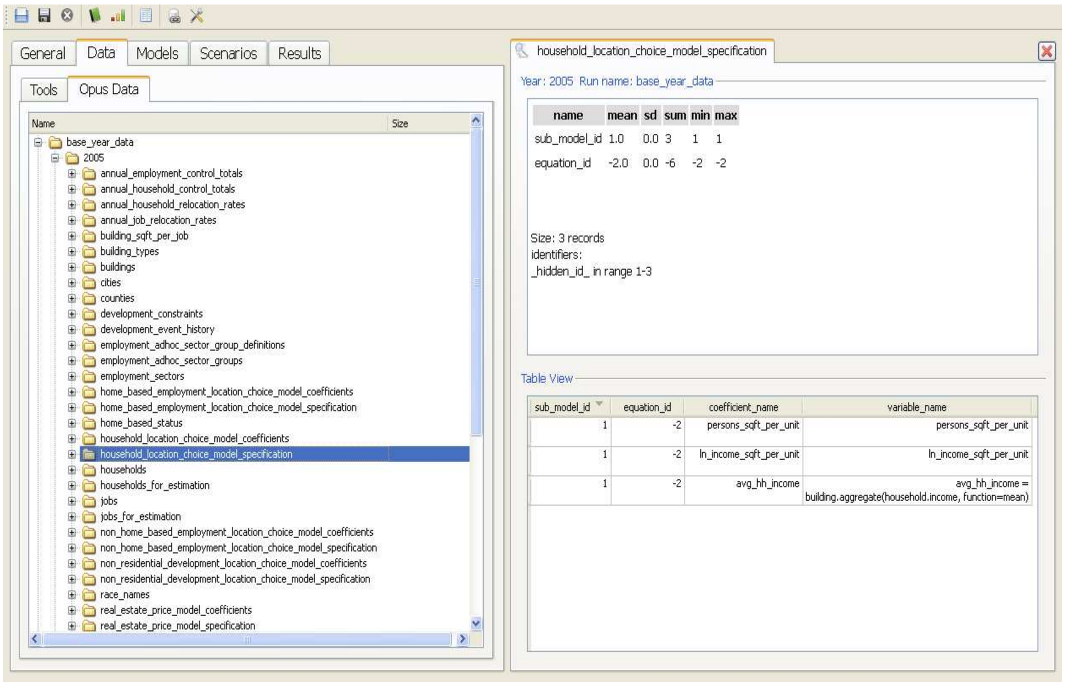


Figure 8: Model specification table in OPUS dataset.

8 Model Simulation

- One can choose the **models that we want to run during the simulation** in the “Scenarios” onglet of the GUI (see Fig. 11), or alternatively in the file `san_antonio_zone.xml`.
- Before running a simulation, one has to check (again in the “Scenarios” onglet of the GUI, or in the file `san_antonio_zone.xml`) that the **set of tables to be cached for the simulation** is correct (see Fig. 12). All the necessary tables have to be selected in the GUI (this is not done automatically by the program) or specified in the file `san_antonio_zone.xml`. One also has to carefully check that all the tables to be cached are effectively available in your dataset. In comparison with the `san_antonio_zone` project, one has to untick the `household_characteristics_for_ht` table, that does not exist for Brussels.
- During the simulation, **the different models are run sequentially** following the list defined in the “Scenarios” onglet of the GUI (see Fig. 13). This order can be modified in the file `san_antonio_zone.xml`.

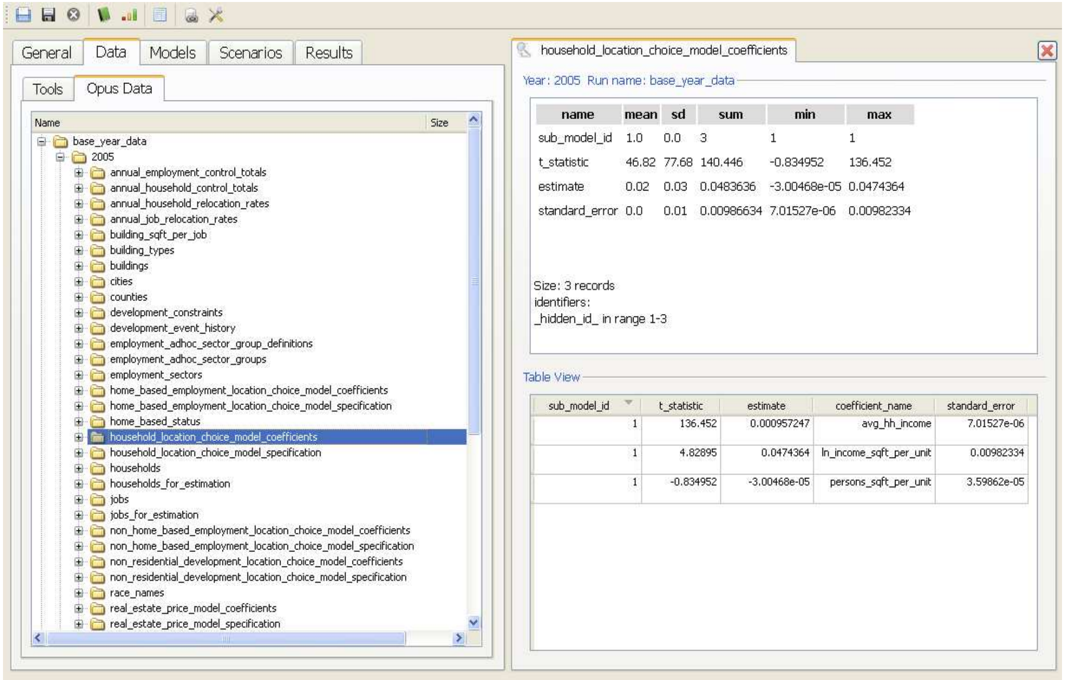


Figure 9: Model coefficients table in OPUS dataset.

The simulation is also done year after year (*i.e.* all the models are simulated for the first year, then all the models are simulated again for the next year and so on). Note that there are hence no potential conflicts between inputs and outputs of the different models. Indeed, all the models are using the tables of the preceding year for the simulation of the current year[5].

- **Some models need other models to have been already run before they can be executed themselves**. These specific prerequisites are defined in `urbansim_zone.xml`, in the third line of the model definitions. This is notably the case for the following models:
  - The `household_relocation` model has to be run priorly to the execution of the `household_location_choice` model. Indeed, it is intuitive to understand that one has first to determine whether a household is intended to move before computing where this house-

[5]To simulate the year 2006, all the models will use the tables available for 2005. Then, to simulate the year 2007, all the models will use the tables created during the simulation for the year 2006. And so on.

Figure 10: `no attribute coefficients` error message.

hold will effectively move.

– Similarly, the `employment_relocation` model has to be run priorly to the execution of the `employment_location_choice` model.

When running the `household_location_choice` model alone for example, we get the following error message (see Fig. 14):

`NoneType attribute`

- **Control totals are** currently **used**, but one also can imagine to use fertility and mortality models instead of that or in addition to that. The `annual_household_control_totals` table must contain, like it is done in the `san_antonio_zone` project, only one total number of households

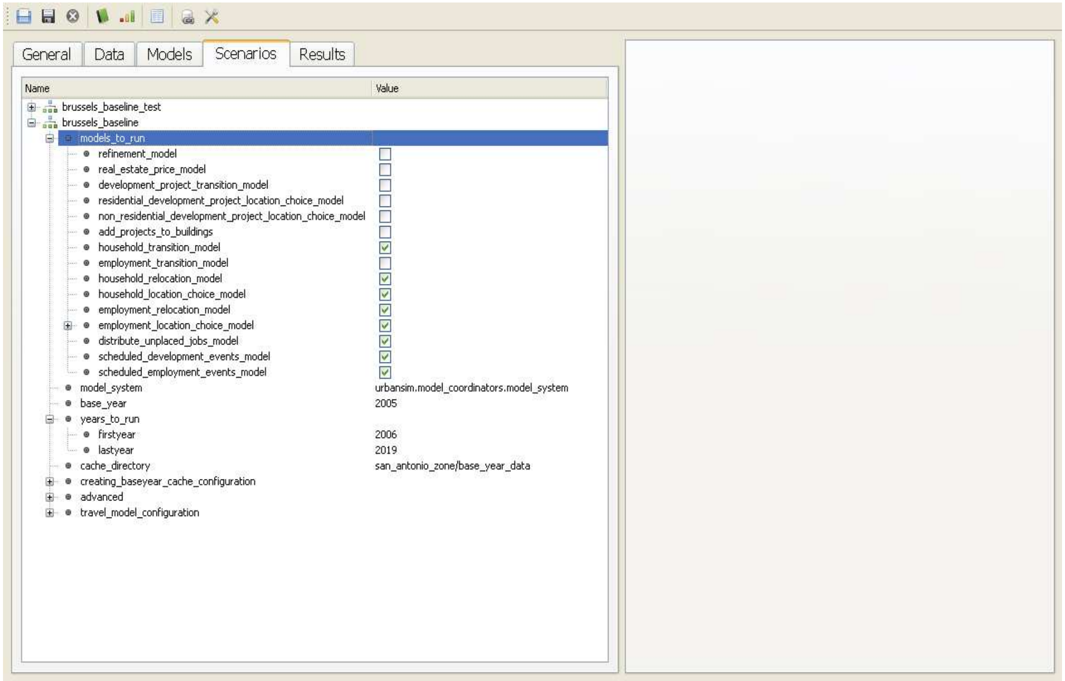


Figure 11: Models to be run during a simulation in OPUS.

for each year. Your dataset must contain the two following tables in order to work correctly with control totals:

- `annual_household_control_totals`
- `annual_employment_control_totals`

- **Simulation results are written automatically in your dataset** (the tables are automatically created by OPUS) (see Fig. 15).

- One can **run an indicator** (*c.f.* Section 6) **to visualize the** obtained **results** (see Fig. 16). As already mentioned, an indicator can be created in the same one creates a variable.

- An example of simulation results is given in Fig. 17.

- It might happen that your computer gets **out of memory** during the simulation. In that case you get the following error message (see Fig. 18):

```
MemoryError
```

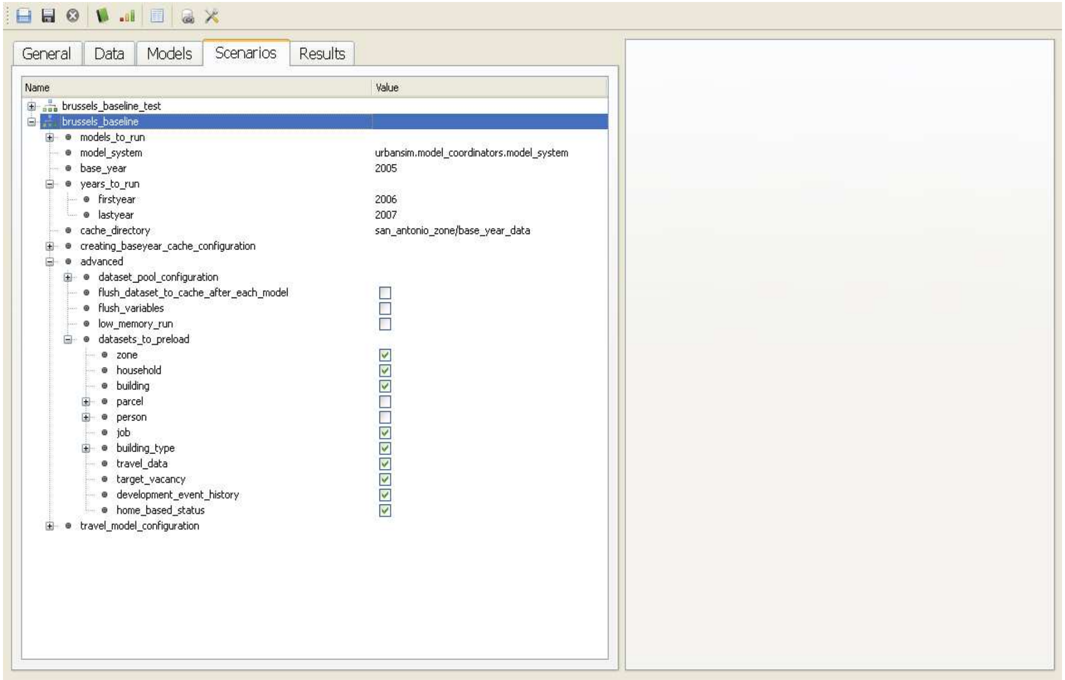


Figure 12: Tables to be cached during a simulation in OPUS.

9 Some Hints to Proceed with the New Data for Brussels

- Start from the `san_antonio_zone` example (or you can also create a model from template, the zone version, and take inspiration on the `san_antonio_zone` example thereafter)
  - Open the `san_antonio_zone` example and save the project as `brussels_zone.xml` (a file `brussels_zone.xml` will be created in `C:\opus\project_configs`)
  - Create a file `brussels_zone.xml` in:

    `C:\opus\src\brussels_zone\configs`

    To do this (and to build the mandatory source project package), create first a folder `brussels_zone` in:

    `C:\src`

    For that, open Python and type:

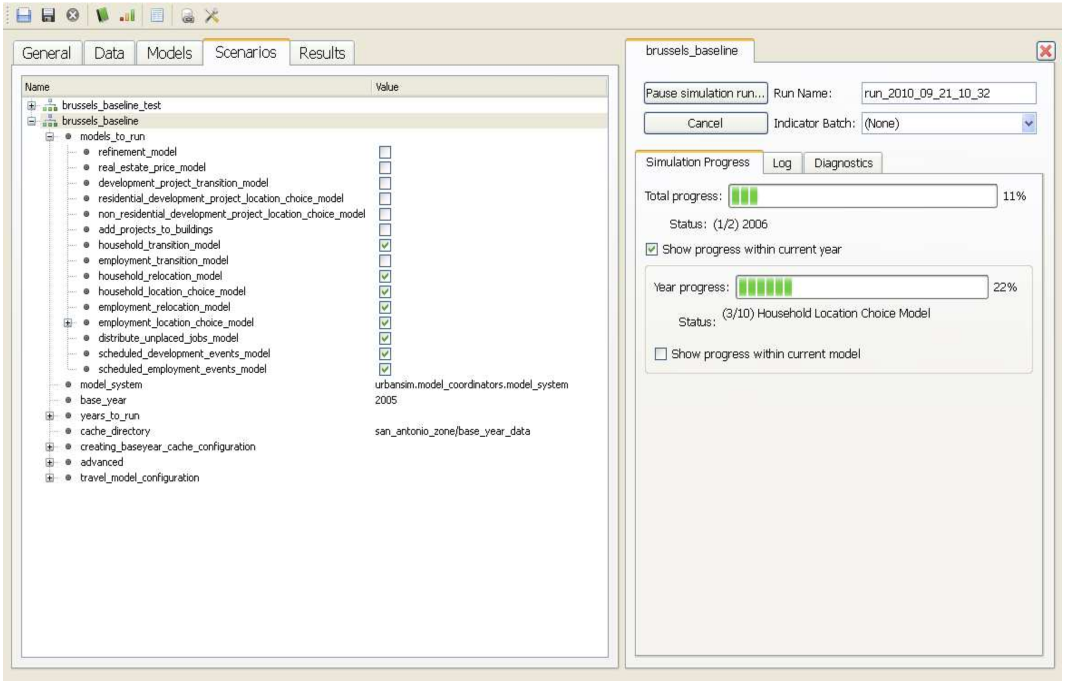


Figure 13: Sequential run of the models during a simulation in OPUS.

```
from opus_core.opus_package import create_package
```

then

```
create_package('C:/opusworkspace', 'brussels_zone')
```

Then, the folder `brussels_zone` that has been created in the `C:/opusworkspace` folder should be moved and placed into the directory `C:\opus\src`. If not already created, create a `configs` folder in `C:\src\brussels_zone`. Copy the files `baseline.py` and `baseline_extimation.py` from `C:\src\san_antonio_zone\configs` into `C:\src\brussels_zone\configs`. Finally, create (if not already created) a file `brussels_zone.xml` (by copying the corresponding file of the `san_antonio_zone`, the `eugene_zone` or the `psrc_zone` examples) into the folder `C:\src\brussels_zone\configs`.

– Create a folder in `C:opus/data` named `brussels_zone` inside which (`opus/data/brussels_zone`) you will be able to place the (consistent) `base_year_data` folder that you have constructed for Brussels (containing your `brussels` data in OPUS cache format, *i.e.* `.li4`, `.li6`, `.li8`, `.is1` or `.lf4` files)

Figure 14: `NoneType attribute` error message.

  – Open the `brussels_zone.xml` files (for example with Eclipse) and change all references to `san_antonio_zone` by `brussels_zone` (*c.f.* Report on UrbanSim by Peter Goodings Swartz, 2008, pages 10 and 11)
  – Then one has to create a coherent datasaset and possibly adapt the models...

10 List of the Models That Work

- `household_transition` model:
  Adds new households or removes households to match control totals. This model uses the `annual_household_control_totals` table, that

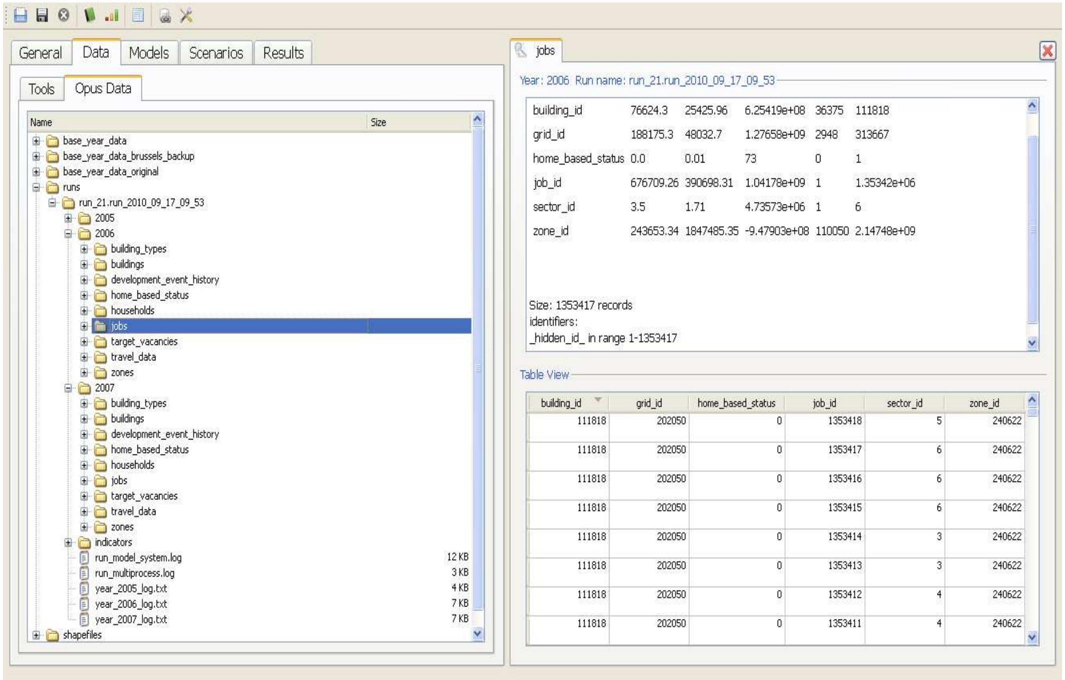


Figure 15: Simulation results written in OPUS dataset.

must be present in your dataset.

- `household_relocation` model:
  Predicts households decision to relocate within the city. This model uses the `annual_household_relocation_rates` table, that must be present in your dataset.

- `household_location_choice` model:
  Predicts location choices for new or moving households. This model is estimated over the `households_for_estimation` table.

- `employment_transition` model:
  Adds new jobs or removes jobs to match control totals. This model uses the `annual_employment_control_totals` table, that must be present in your dataset.

- `employment_relocation` model:
  Predicts job (employer) decision to relocate within a region. This model uses the `annual_job_relocation_rates` table, that must be present in your dataset.

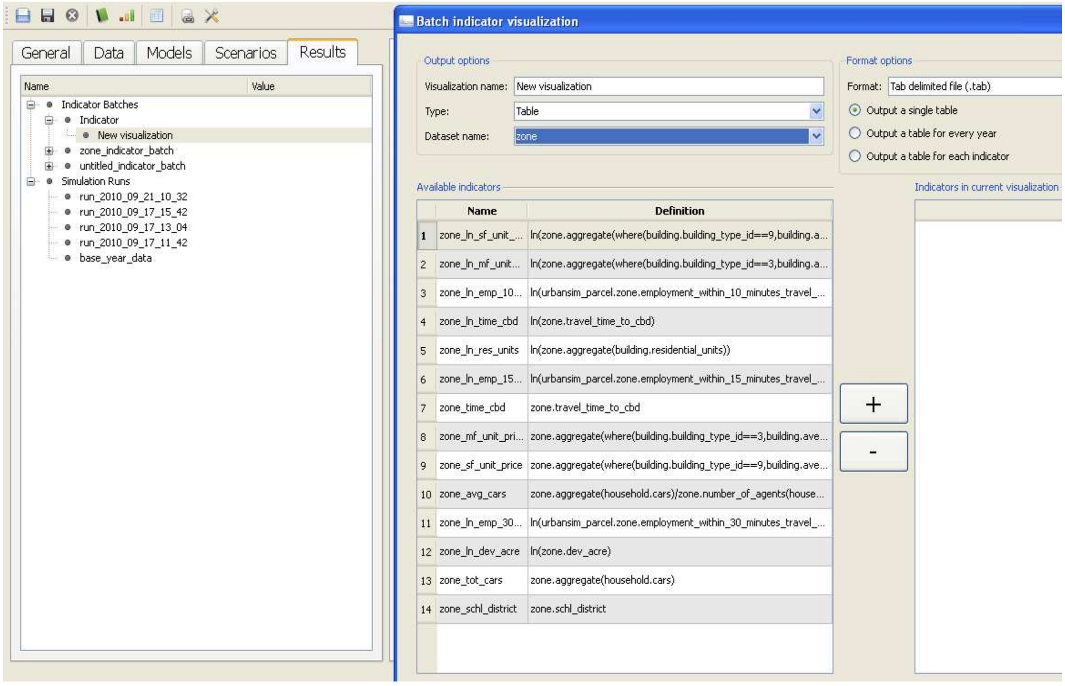


Figure 16: Creation of an indicator batch in OPUS.

- `employment_location_choice` model:
  Predicts location choices for new or moving jobss. This model is estimated over the `jobs_for_estimation` table.

- `distribute_unplaced_jobs` model:
  Allocates sectors of employment proportionally. This model works on the `jobs` table.

- `scheduled_development_events` model:
  Handles and executes scheduled development events. This model uses the `scheduled_development_events` table, that is present in your dataset.

- `scheduled_employment_events` model:
  Handles and executes scheduled employment events. This model uses the `scheduled_employment_events` table, that is present in your dataset.

zone_table-1_2005_zone__zone_tot_persons

zone_id	zone_tot_persons_2005
110050	14967.0
110370	13214.0
120050	12025.0
120090	14580.0
120250	70567.0
120290	14047.0
120300	14255.0
120340	6559.0
120350	17022.0
120400	20699.0
210010	89699.0
210020	29270.0
210030	18187.0
210041	40666.0
210042	16397.0
210043	57617.0
210044	21977.0
210045	10342.0

zone_table-1_2006_zone__zone_tot_persons

zone_id	zone_tot_persons_2006
110050	14256.0
110370	13208.0
120050	13256.0
120090	14788.0
120250	70211.0
120290	14910.0
120300	15034.0
120340	6807.0
120350	18071.0
120400	20356.0
210010	85418.0
210020	28566.0
210030	17332.0
210041	37758.0
210042	15329.0
210043	54006.0
210044	21328.0
210045	10045.0

zone_table-1_2007_zone__zone_tot_persons

zone_id	zone_tot_persons_2007
110050	14316.0
110370	13570.0
120050	13504.0
120090	14947.0
120250	70846.0
120290	15150.0
120300	15337.0
120340	6944.0
120350	18215.0
120400	20536.0
210010	85613.0
210020	28693.0
210030	17325.0
210041	37772.0
210042	15344.0
210043	54217.0
210044	21524.0
210045	10101.0

Figure 17: Example of OPUS simulation results (here the indicator is the total number of persons per zone).

11 What Still Does Not Work

(a) Development Project Location Choice Models

- This concerns three models that are very closely related:
  - `development_project_transition` model:
    Predicts new development projects to be located.
  - `residential_development_project_location_choice` model:
    Predicts locations for new residential development projects.
  - `non_residential_development_project_location_choice` model:
    Predicts locations for new non-residential development projects.
- These models are dependent on the specific `building_types` that you consider in your project (*i.e.* each created development project will involve a given `building_type`).
- In the old dataset available for Brussels, there was an almost complete lack of data concerning the tables to be used by these models.

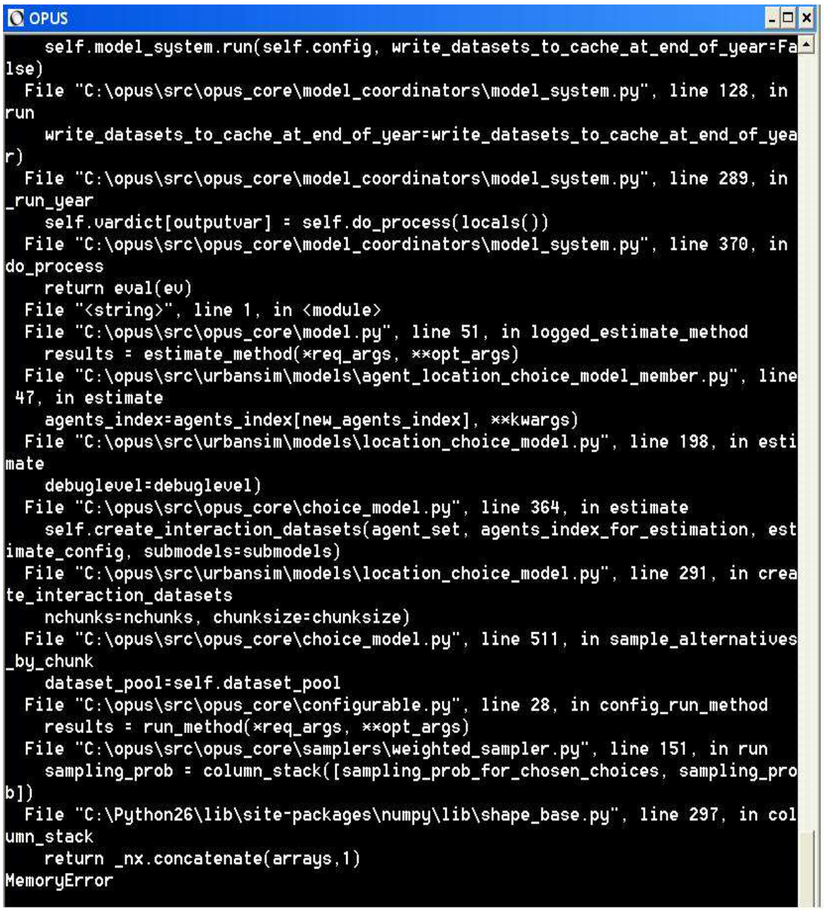


Figure 18: `MemoryError` error message.

- Actually, both the `residential_development_project_location_choice` model and the `non_residential_development_project_location_choice` model are estimated on the `development_event_history` table (and not on the `scheduled_development_events` table). This table must contain **enough data for all types of building to be estimated** in a dedicated submodel. One has to be very careful when creating this `development_event_history` table since it involves several different `xxx_id` entries and it is thus hard to remain consistent with respect to all other related tables.

- To allow some residential development projects to take place, some buildings must have a `residential_units_capacity` greater than their current number of `residential_units`. The same is true for non residential development projects. One has thus to carefully check the `buildings` table regarding that.

(b) Real Estate Price Model

- The `real_estate_price` model predicts price per unit for each building (*i.e.* it is hence highly more disaggregated than if it was computing the price per unit for each building type!).
- The problem occurs here because we have specific `building_types` in our project (that are different from the ones in the `san_antonio_zone` project) and for that reason the submodels that are constituting the `real_estate_price` model have to be modified in order to cope with our particular `building_types`.
- In that regard, we have tried to simplify or to remove some of the submodels that were composing the `real_estate_price` model to cope exactly with the Brussels case and to its specific `building_types`. However, some specificities of the `san_antonio_zone` project, probably hard coded, still remain... And we get thus the following error message when trying to estimate the `real_estate_price` model (see Fig. 19):

  ```
  different dataset names for variable and dataset
  ```

- Starting from scratch in order to create a dedicated `real_estate_price` model is even more messy...
- Note that the tables concerned for the estimation of the `real_estate_price` model are: `buildings` and `building_types`.
- The outcome of the model is the following: `building.average_value_per_unit`.
- The same kind of problem has also appeared in the other case studies (Zurich, Paris). It is not surprising since each of these projects is also likely to have its own specific `building_types`.

12 What Remains to Do with the New Data

- **Results visualization**: while the results can now only be visualized as tables, the creation of shapefiles (corresponding to the zones that will be defined with the new dataset for Brussels) to be included in

  `C:\opus\data\brussels_zone\shapefiles`

Figure 19: `different dataset` error message.

will allow to see the simulation results as graphical (colored) maps and/or animated maps.

- **Integration of a MatSIM add-on** in order to incorporate more precise and fully disaggregated transportation features[6]. This integration is far from being straightforward since:
  - UrbanSim is a Python based software while MatSIM is Java based.

[6]In UrbanSim, the transportation processes are mostly static. Indeed, the travel times are fixed and are given for each trip between two zones, but they do not take into account potential congestion effects that might arise when too many people are traveling on the same route at the same time (*i.e.* contrary to the rest of the software that includes fully microsimulated processes, the transportation aspects in UrbanSim are in a sense aggregated).

– The communication and the data transfer between the two softwares during simulation will hence reveal themselves to be tricky.

Appendix 1: Some Words Concerning the Files Provided in the DVD

The provided DVD contains the following folders:

- **Brussels San Antonio Minimal Set Of Files**:
  The minimal set of files needed to run the Brussels dataset inside the `san_antonio_zone` project (*i.e.* the project name remains `san_antonio_zone` but the Brussels data are used inside it). This is composed of the following files or folders:

  – The `san_antonio_zone.xml` file to be placed in

    `C:\opus\project_configs`

  – The `urbansim_zone.xml` file to be placed in

    `C:\opus\src\urbansim_zone\configs`

  – The `base_year_data` folder to be placed in

    `C:\opus\data\san_antonio_zone`

  – The `runs` folder to be placed in

    `C:\opus\data\san_antonio_zone`

- **Brussels Minimal Set Of Files**:
  The minimal set of files needed to run the Brussels dataset as a specific `brussels_zone` project[7]. This is composed of the following files or folders:

  – The `brussels_zone.xml` file to be placed in

    `C:\opus\project_configs`

  – The `urbansim_zone.xml` file to be placed in

    `C:\opus\src\urbansim_zone\configs`

[7] Compared to running the Brussels dataset inside the `san_antonio_zone` project, running the dataset as a specific `brussels_zone` project leads to a `MemoryError` (not enough memory) when estimating the `household_location_choice` model. Except that, everything seems to run similarly (estimation and simulation).

- The `brussels_zone` folder to be placed in

  `C:\opus\data`

- **Brussels Original Data**:
  The dataset received originally from Zachary. The data are provided in `.txt` or `.csv` format, and a `base_year_data` folder to be used in UrbanSim, created with these data, is also given.

- **Brussels Updated Data**:
  The updated dataset for Brussels, where several tables have been modified and/or created in order to get a consistent and usable dataset. In more details, this folder contains the following subfolders:

  - **Base Year Data Folder**:
    The `base_year_data` folder to be used in UrbanSim containing the updated dataset.
  - **MySQL Data**:
    The content of the MySQL database that contains the updated dataset.
  - **New Updated Data Computation**:
    The Matlab and Excel files that have been used to create and/or modify the updated data.
  - **New Updated Data Text Files**:
    The updated dataset in `.txt` format.

- **UrbanSim Installation**:
  All the necessary files for the installation of OPUS on Windows XP SP2. In more details, this folder contains the following subfolders:

  - **MySQL - Navicat**:
    Installation files for setting up a database (MySQL Server and Navicat Lite).
  - **New Examples**:
    Files received at the UrbanSim Zurich Tutorial concerning additional examples of projects (`san_antonio_zone`, `psrc_parcel`, `durham_zone`).
  - **OPUS**:
    OPUS 4.3.0 installer file.
  - **Tortoise**:
    Tortoise 1.6.12 installer file.

- **Presentation**:
  Final version of the September 21st presentation and the corresponding latex files. Snapshots of different errors that might occur in UrbanSim and several snapshots of the software running.

- **Report**:
  Final version of the report and the corresponding latex files.

Appendix 2: Installation of the Older Version 4.2.2 of UrbanSim

In the following, we briefly present the main steps to install the older version 4.2.2 of UrbanSim on Windows XP SP2.

- **Install the latest stable version** of the Windows Installer **of UrbanSim** (on April 2010, it was done through Opus 4.2.2), available at:

  `http://www.urbansim.org/Download/WindowsInstaller`

  The file should be named `opus_installer.exe`.

- Although it should already have been done by the WindowsInstaller, you have to **install** once more and properly **MySQL** on your computer. To do this, you must follow precisely the instructions given at (see Section 3):

  `http://www.urbansim.org/Download/InstallingMySQL`

  During the installation process of MySQL, you will have to choose a password for MySQL that we will denote here by `password_1`. At the end of the installation process, you have to **enter precisely the three following commands in the MySQL Command Line Client** (Start/All Programs/MySQL/MySQL Server 5.1/MySQL Command Line Client):

  * `GRANT create, delete, drop, index, insert, select, update, alter, create temporary tables, file, reload ON *.* TO urbansim@localhost IDENTIFIED BY 'password_1';`

* SET PASSWORD FOR 'urbansim'@'localhost' = OLD_PASSWORD('password_1');

* FLUSH PRIVILEGES;

The above three command lines will allow UrbanSim to access the MySQL database. The chosen password for MySQL (*i.e.* `password_1`) has still to be correctly indicated to UrbanSim (it is mandatory since UrbanSim will thereafter access to the MySQL database for data importation and exportation). This can be done either by editing the following `.xml` file:

`C:\opus\settings\database_server_configurations.xml`

or through the GUI interface where one of the buttons ('`Open Database Connection Settings`') on the main panel allows us to do so. We have here to modify the following database connection:

`mysql_test_database_server`

(this database will be used thereafter in the `csv_to_sql` and `sql_data_to_opus` tools). For that particular database connection, the host_name has to be set to `localhost` and the password to `password_1` (the user_name remains `urbansim`). After editing the `.xml` file, one must finally have:

```
<mysql_test_database_server setexpanded="True">
<protocol choices="postgres|mssql|mysql|sqlite"
type="string">mysql</protocol>
<host_name type="string">localhost</host_name>
<user_name type="string">urbansim</user_name>
<password type="password">password_1 </password>
</mysql_test_database_server>
```