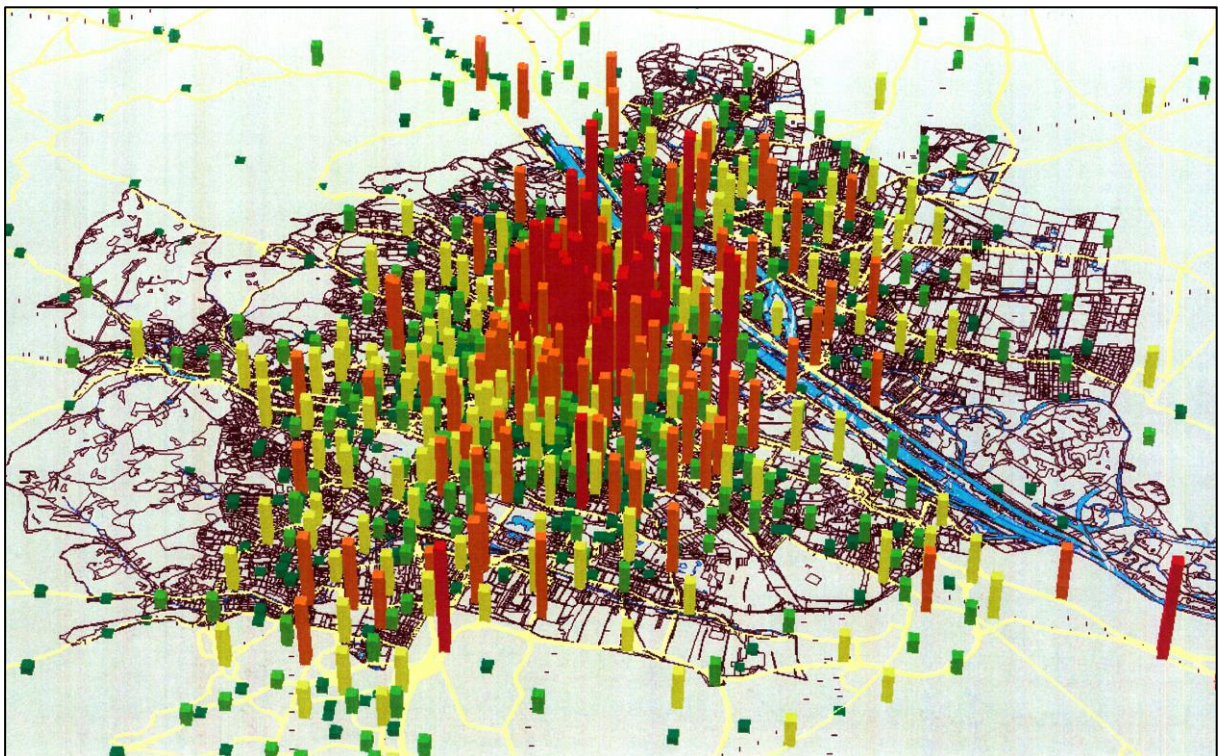


---

# CityGML 3D City Database Simulation Package PostgreSQL Version

## Documentation

Last update: 15 May 2018





## Active participants in development

Name	Institution	Email
Giorgio Agugiaro	AIT – Austrian Institute of Technology G.m.b.H. Center for Energy – Smart Cities & Regions	<a href="mailto:giorgio.agugiaro@ait.ac.at">giorgio.agugiaro@ait.ac.at</a>
Edmund Widl	AIT – Austrian Institute of Technology G.m.b.H. Center for Energy – Electric Energy Systems	<a href="mailto:edmund.widl@ait.ac.at">edmund.widl@ait.ac.at</a>

## Acknowledgements

The 3D City Simulation Package (PostgreSQL version) was developed in the framework of the following international project:

The JPI Urban Europe, ERANET Co-fund Smart Cities/Urban Futures 2015 “**IntegrCiTy**” project (FFG Project number 855078), funded (in Austria) by the Austrian Ministry for Transport, Innovation and Technology and with the support from the European Union’s Horizon 2020 research and innovation programme.



Homepage: <http://iese.heig-vd.ch/projets/integrcity>

Finally, the authors would like to thank **Pablo Puerto** (CREM) for the fruitful discussions.

The image on the front page is courtesy of Jan Peters-Anders (AIT).



# Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>9</b>
	<i>1.1 System and design decisions.....</i>	<i>9</i>
<b>2</b>	<b>DATA MODELLING.....</b>	<b>10</b>
<b>3</b>	<b>DATABASE DESIGN .....</b>	<b>12</b>
	<i>3.1 Database schema of the Simulation Package.....</i>	<i>12</i>
	3.1.1 Sequences .....	13
	3.1.2 Stored procedures .....	14
	3.1.3 Views .....	15
<b>4</b>	<b>TEST DATA.....</b>	<b>17</b>
<b>5</b>	<b>INSTALLATION.....</b>	<b>18</b>
	<i>5.1 System requirements.....</i>	<i>18</i>
	<i>5.2 Automatic installation .....</i>	<i>18</i>
	<i>5.3 Manual installation .....</i>	<i>20</i>
	<i>5.4 Installation of the test data.....</i>	<i>21</i>
<b>6</b>	<b>DOCUMENT REVISION NOTES .....</b>	<b>23</b>
	<b>NOTES .....</b>	<b>25</b>



## Disclaimer

The *3D City Database Simulation Package*, developed by Austrian Institute of Technology, Center for Energy, Smart Cities and Regions Research Field (AIT), is free software and licensed under the Apache License, Version 2.0. See the file LICENSE file shipped together with the software for more details. You may obtain a copy of the license at <http://www.apache.org/licenses/LICENSE-2.0>.

THE SOFTWARE IS PROVIDED BY AIT "AS IS" AND "WITH ALL FAULTS." AIT MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE QUALITY, SAFETY OR SUITABILITY OF THE SOFTWARE, EITHER EXPRESSED OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

AIT MAKES NO REPRESENTATIONS OR WARRANTIES AS TO THE TRUTH, ACCURACY OR COMPLETENESS OF ANY STATEMENTS, INFORMATION OR MATERIALS CONCERNING THE SOFTWARE THAT IS CONTAINED ON AND WITHIN ANY OF THE WEBSITES OWNED AND OPERATED BY AIT.

IN NO EVENT WILL AIT BE LIABLE FOR ANY INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES HOWEVER THEY MAY ARISE AND EVEN IF AIT HAVE BEEN PREVIOUSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.





# 1 Introduction

A crucial prerequisite for the creation of meaningful simulations model is the availability of high-quality data. Within this context, the CityGML data model and its domain extensions (e.g., the Energy ADE and the Utility Network ADE) enable a coherent approach for storing geospatial and semantic information for further use in modelling and simulation. However, on top of the domain-specific data stored with the help of these data models, additional meta-information is required to execute an actual simulation. This is especially true in the case of co-simulation, where not only individual simulator configurations are required (e.g., integrator steps sizes or initial conditions), but also information regarding the coupling and orchestration of several simulator instances. Consequently, the logical next step is a persistency schema for this type of information that integrates into the CityGML framework.

The 3DCityDB Simulation Package is – hopefully – the first step into this direction. The wish is that the availability of a common, shared database schema will contribute and foster adoption and further improvement of the Simulation Package in a sort of virtuous circle. Hence, any feedback, constructive suggestions and help to correct, improve and test the current implementation are greatly welcome.

## 1.1 System and design decisions

Currently, the 3D City Database schema is provided for Oracle with the Spatial or Locator licensing option (10G R2 or higher) and PostgreSQL (9.1 or higher) with PostGIS extension (2.0 or higher). The 3DCityDB extension for the Simulation Package requires the latest version of the 3DCityDB (v. 3.3.2) and is currently implemented for PostgreSQL only. As the overall goal of this extension is to facilitate direct connection and usage of the 3DCityDB relational database by users and applications programmers, a number of design and implementation decisions were taken according to the inspiring criteria briefly listed in the following:

- a) Follow the Simulation Package data model as close as possible. This means also that tables and attribute names in the database are kept as close as possible to the original names of the Simulation Package data model as indicated in the UML diagrams;
- b) Follow the conventions adopted for the 3DCityDB documentation when writing this document, in order to guarantee a certain layout and visual coherency.

For these reasons, several concepts will be only briefly mentioned in this document, while providing a reference to the existing documentation. The reader is therefore encouraged to keep at least a copy of the 3DCityDB and of the Simulation Package documentation at hand.

The **3DCityDB documentation** can be accessed on-line at this URL:

<https://github.com/3dcitydb/3dcitydb/tree/master/Documentation>

## 2 Data Modelling

The following pages cite several parts of the of CityGML specification which are necessary for a better understanding. Design decisions in the model are explicitly visualised within the UML diagrams. In addition, whenever necessary, parts of the CityGML UML diagrams are also depicted, in order to better show where the Simulation Package elements connect to the existing classes.

For intuitive understanding, classes that will be merged to a single table in the relational schema are shown as **orange blocks** in the UML diagrams.

The UML diagram of the Simulation Package is presented in Figure 1. The package's structure specifically targets co-simulation setups. At the same time, the design aims to be as generic as possible, enabling the application of the Simulation Package to a large variety of simulations tools and co-simulation environments.

To this end, the following classes have been defined:

Instances of class **Simulation** are – from a hierarchical point of view – the top-level objects describing a co-simulation setup. It links all the relevant entities that are required to define the composition of several simulators (see definition of class **Node** and **PortConnection** below). Furthermore, it provides the possibility to store parameters related to the orchestration of the simulators (i.e., configuration parameters for the co-simulation master algorithm) as generic attributes (contained in class **GenericParameter**). In addition, class **Simulation** references class **Scenario** from the CityGML Scenario ADE (please refer to the Scenario ADE documentation<sup>1</sup> for further details).

Instances of class **Node** represent the basic simulation units of a co-simulation setup. Nodes are an abstraction of the simulation models and tools themselves, providing the information relevant to the initialization and execution of the simulation units. Information about the associated simulation model (e.g. name or parameters) can be specified as generic attributes (contained in class **GenericParameter**). Input and output variables are represented by instances of class **Port**, whereas specific information about the associated simulation tool is stored in an instance of class **SimulationTool**. Furthermore, nodes can be linked with CityGML objects, which allows to link the otherwise generic node objects to domain-specific semantic data (useful for instance for automated model creation or validation).

The **AbstractPort** class is an abstract class that is implemented by **InputPort** and **OutputPort**. Instances of class **InputPort** represent the input variables of a simulation node that can be used by the co-simulation master to send information to the associated simulation model/tool. Likewise, instances of class **OutputPort** represent the output variables of a simulation node that can be used by the co-simulation master to retrieve information from the associated simulation model/tool. In both cases, a port is intended to represent only a single scalar variable and must correspond to a variable in the associated simulation model with the

---

<sup>1</sup> [https://github.com/gioagu/3dcitydb\\_ade/tree/master/04\\_scenario\\_ade/manual](https://github.com/gioagu/3dcitydb_ade/tree/master/04_scenario_ade/manual)

same name and type. Like in the case of class Node, ports can be associated to CityGML objects, linking the otherwise generic port objects to domain-specific semantic data.

The **ConnectionPort** class provides the possibility to link ports from different nodes, with each instance linking exactly one input port with one output port (corresponding to the exchange of one scalar value between these two ports). Together with the class Node, this class is the fundamental building block for defining the coupling between simulation models/tool in a co-simulation setup.

The **SimulationTool** class allows to attach information specific to the simulation tool used by a node. This information is stored separately from the simulation nodes in order to avoid duplications, because in a co-simulation several nodes may use instances of the same simulation tool. In order to be flexible enough, the class mostly relies on generic attributes (contained in class GenericParameter) next to a few basic attributes (regarding mainly dependencies on the OS, version, etc.).

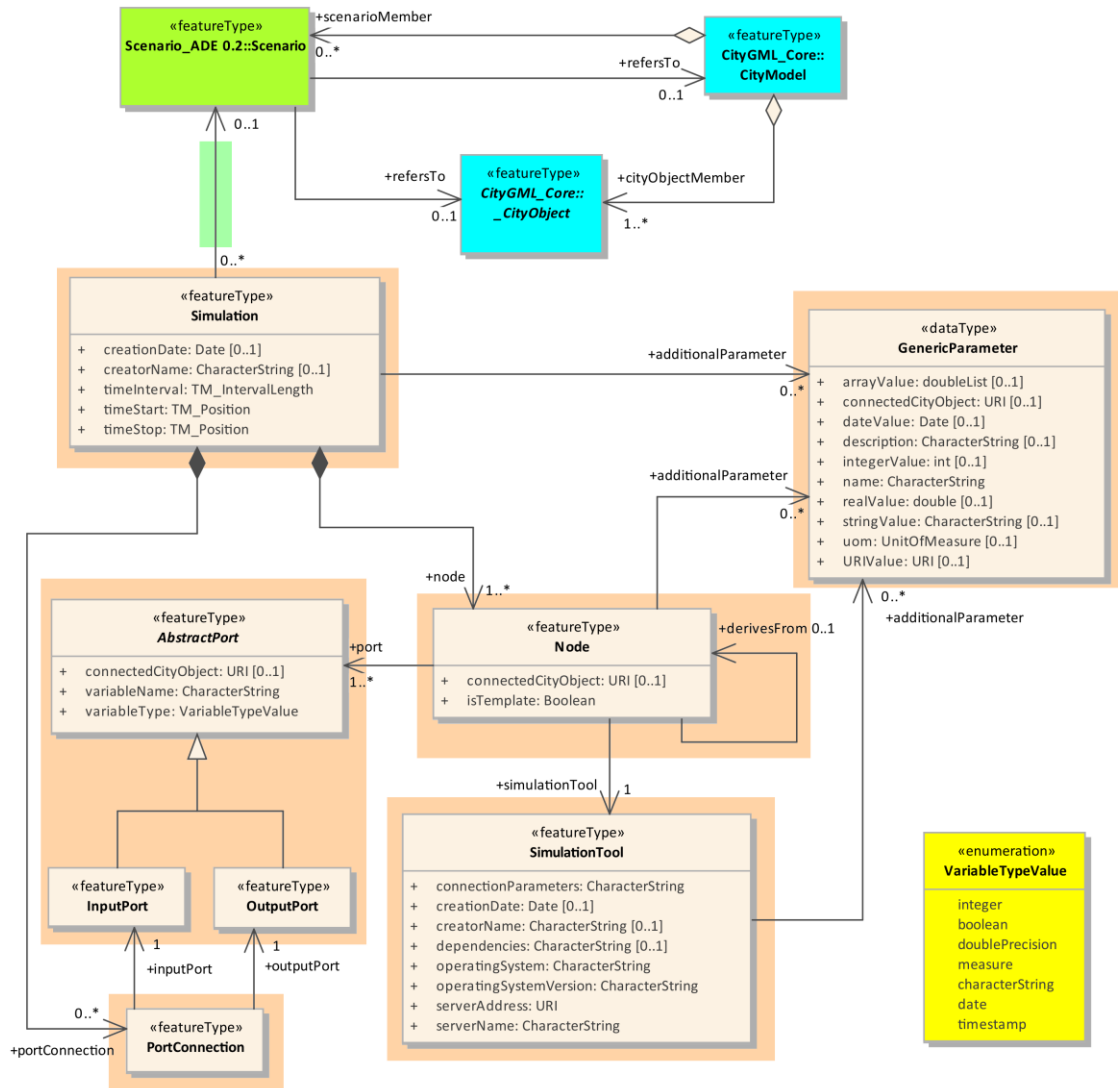


Figure 1: UML diagram of the Simulation Package

## 3 Database design

This section presents and describes the implementation of the Simulation Package as a relational database schema for PostgreSQL.

### 3.1 Database schema of the Simulation Package

In this section, the ER diagram of the database schema of the Simulation Package is shown. The tables correspond to the classes shown in the UML diagram in Section 2. The database relations correspond to the Simulation Package v. 0.1.

In general, the names of the tables correspond the names of the classes. The names of the columns correspond to the names of the attributes (or are as close as possible to the original ones).

Given the quite straightforward mapping between the object-orientated and the entity-relation model, only few explanation notes are added in the following.

In particular, table **PORT** corresponds and merges classes InputPort and OutputPort (and the abstract super class AbstractPort). In order to distinguish between the two classes, the TYPE attribute is used. The attribute TYPE must contain either the “*input*” or the “*output*” value, and it is constrained to be NOT NULL.

Table **GENERIC\_PARAMETER** corresponds to class GenericParameter. The attribute ARRAYVAL is of type numeric[]. Please note that, in order to input an array in PostgreSQL, two main methods exist. The values can be input as a string of comma-separated values between two curly braces (e.g. {1, 3, 5, 6, 7, 2}) or using the PostgreSQL array constructor ARRAY (e.g. ARRAY[1, 3, 5, 6, 7, 2])<sup>2</sup>. The foreign keys SIMULATION\_ID, NODE\_ID, TOOL\_ID reference the tables SIMULATION, NODE and TOOL, respectively.

---

<sup>2</sup> <https://www.postgresql.org/docs/9.1/static/arrays.html>

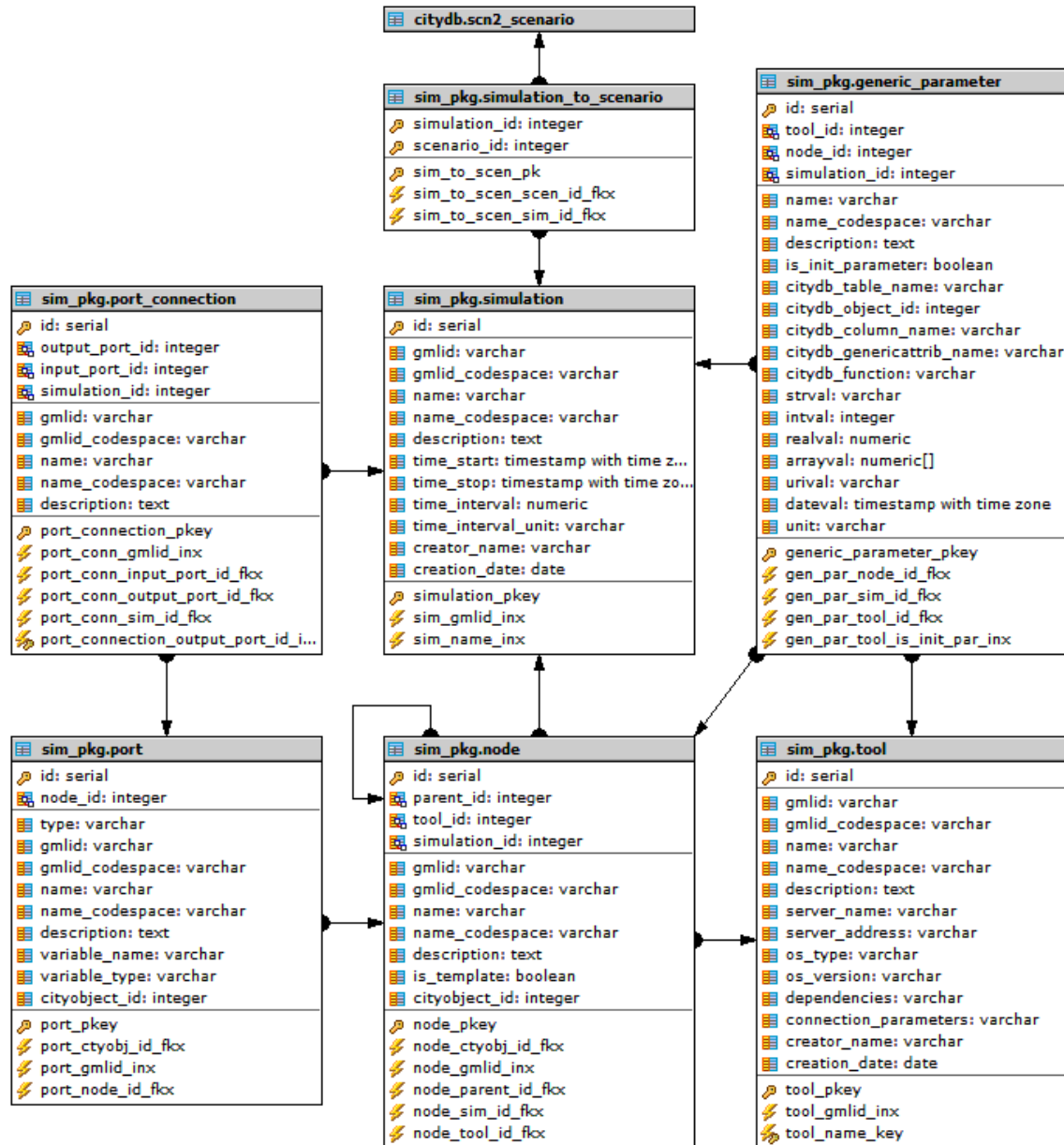


Figure 2: ER diagram of the Simulation Package

### 3.1.1 Sequences

Upon installation, a number of database sequences is also generated and stored in the *sim\_pkg* schema. Analogously to other 3DCityDB tables, it is highly recommended to generate ID values for the respective Simulation Package tables by using the predefined sequences only.

Please note that all new sequences are generated directly within the CREATE TABLE statement as follows:

```
CREATE TABLE sim_pkg.simulation (
  id serial PRIMARY KEY,
  name varchar,
  ...);
```

As a matter of fact, the automatically created sequence corresponds to the statement

```
CREATE SEQUENCE simulation_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
ALTER TABLE simulation_id_seq OWNER TO postgres;
```

Although theoretically equivalent, a practical difference has been found so far. Safe Software’s FME PostGIS and PostgreSQL writers can profit of the auto-increment property of serial types in INSERT statements only if a sequence is created with a CREATE TABLE statement. Please read the last answer in <https://knowledge.safe.com/questions/1703/serial-data-types-and-insert-into-postgresql.html>

### 3.1.2 Stored procedures

The 3DCityDB Simulation Package is shipped with a set of stored procedures (or “functions”, in the PostgreSQL jargon). They are automatically installed during the setup. In the PostgreSQL version, stored procedures are written in PL/pgSQL and stored in the database schema *sim\_pkg*. Please note that, as already mentioned, the input parameters are omitted and indicated by a simple “...” in the following text for better readability.

A “delete” stored procedure allows to delete a Simulation Package object and all its dependencies at once. All stored procedures of this kind are named `delete_*(...)`, where *\** is generally the corresponding name of the object to be deleted from the database. So, `delete_simulation(simulation_id)` will delete all data related to the simulation having as ID the `simulation_id` (e.g. 123). Conceptually, the delete stored procedures reflect the analogous ones shipped with the “vanilla” 3DcityDB.

In Table 1 all stored procedures shipped with this implementation of the Simulation Package are listed.

DELETE STORED PROCEDURES
<code>delete_generic_parameter(...)</code>
<code>delete_node(...)</code>
<code>delete_port(...)</code>
<code>delete_port_connection(...)</code>
<code>delete_simulation(...)</code>
<code>delete_tool(...)</code>
<code>delete_generic_parameter(...)</code>
INSERT STORED PROCEDURES
<code>insert_generic_parameter(...)</code>
<code>insert_node(...)</code>
<code>insert_port(...)</code>
<code>insert_port_connection(...)</code>

insert_simulation(...)
insert_tool(...)
<b>MISCELLANEOUS STORED PROCEDURES</b>
cleanup_schema(...)

**Table 1: Stored procedures shipped with this implementation of the Simulation Package. All stored procedures are stored in schema *sim\_pkg***

### 3.1.3 Views

Data belonging to a specific class can be stored in different tables in the database. As a matter of fact, the database structure can be sometimes complex in terms of CRUD<sup>3</sup> operations (i.e. select, insert, update and delete)..

For this reason, and in order to facilitate CRUD operations at database level, some views are also provided. Views represent a facilitated way to access data, ideally providing one single table that “hides” the complex structure of the actual database. Moreover, if a view is also updatable, entries can be updated, inserted and deleted directly interacting with the view. A series of triggers and trigger functions must be implemented therefore for each view.

All views are installed in the database schema named *sim\_pkg*.

**Please note:** in order to access *any* database object in the *sim\_pkg* schema, **qualified names**<sup>4</sup> consisting of the schema name and the object name separated by a dot **must be used**. This is a design decision. As a matter of fact, the *search\_path* variable of the database instance remains unchanged to the default values (i.e. *search\_path* = *citydb*, *citydb\_pkg*, *public*).

Table 2 contains all views shipped with this version of the 3DcityDB Simulation Package.

VIEW	UPDATABLE? <sup>5</sup>
generic_parameter_node	✓
generic_parameter_sim	✓
generic_parameter_tool	✓
node_no_template	✓
node_template	✓
port_connection_ext	

**Table 2: Views shipped with this implementation of the Simulation Package.**

<sup>3</sup> CRUD represents an acronym for the database operations **C**reate, **R**ead, **U**ppdate, and **D**eleate.

<sup>4</sup> For more details, see <https://www.postgresql.org/docs/9.1/static/ddl-schemas.html>

<sup>5</sup> Requires PostgreSQL 9.3 and above





## 4 Test data

A small exemplary data set to be installed as test data is provided. It is intended to populate the tables with a simple yet consistent dataset and to simplify the understanding of the data model and its manipulation (read/write).

## 5 Installation

The 3D City Database Simulation Package comes with a number of SQL scripts for setting up the relational schema in a PostgreSQL/PostGIS database upon which an instance of the 3DCityDB was previously installed. Detailed instruction on how to set up the 3DCityDB are contained in the **3DCityDB documentation**, accessible on-line at this URL:

<https://github.com/3dcitydb/3dcitydb/tree/master/Documentation>

Moreover, a very useful **hands-on tutorial**, where the most important steps to set up the database and use the Importer/Exporter are described, can be retrieved here:

<https://github.com/3dcitydb/tutorials>

The source code and documentation of the **3DCityDB extension of the Simulation Package** for PostgreSQL can be retrieved here:

[https://github.com/gioagu/3dcitydb\\_ade/tree/master/05\\_simulation\\_pkg](https://github.com/gioagu/3dcitydb_ade/tree/master/05_simulation_pkg)

Please follow the instructions on the next pages in order to complete a proper installation.

### 5.1 System requirements

In order to set up the 3DCityDB extension for the Simulation Package, the 3DCityDB v.3.3.0 needs to be previously installed correctly. All system requirements of the 3DCityDB still apply. PostgreSQL is supported from v. 9.1, although v. 9.3 or higher is recommended. PostGIS 2.0 or higher is required.

**Please note:** additionally, the **3DCityDB extension for the Scenario ADE** is required and needs to be previously installed.

### 5.2 Automatic installation

The automatic installation procedure will install all required packages automatically once the connection parameters to the PostgreSQL server have been set. No other interaction with the user is required. In order to carry out the automatic full installation, the user needs to perform the following steps.

#### Step 1 – Identify the owner of the citydb schema

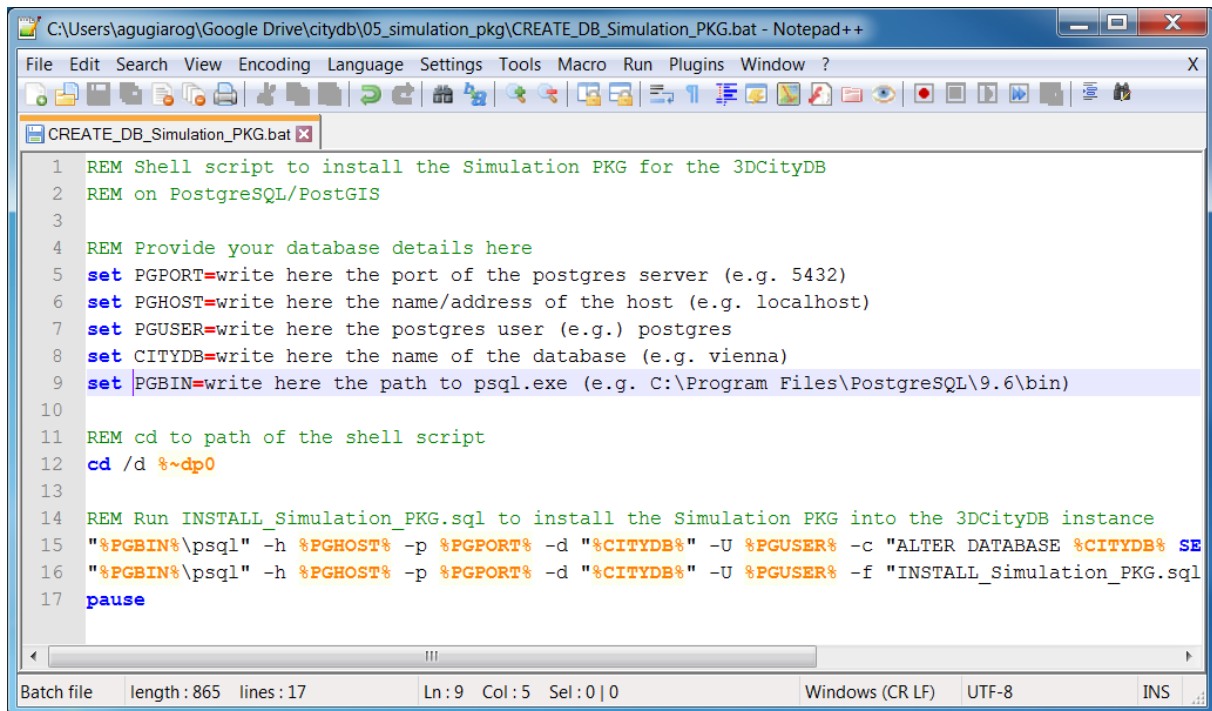
The installation should conveniently be carried out using the same user that installed the *citydb* schema and all included relations.

#### Step 2 – Set the connection parameters and run the installation script

The batch file `CREATE_DB_Simulation_PKG.bat` is located in the folder `\05_simulation_pkg` and can be run from a Windows command shell (or simply by double-clicking it). However, the configuration parameters regarding the PostgreSQL server and the

selected database must be first adapted accordingly. An example is given in Figure 3. For Linux environments, the equivalent file CREATE\_DB\_Simulation\_PKG.sh is also provided.

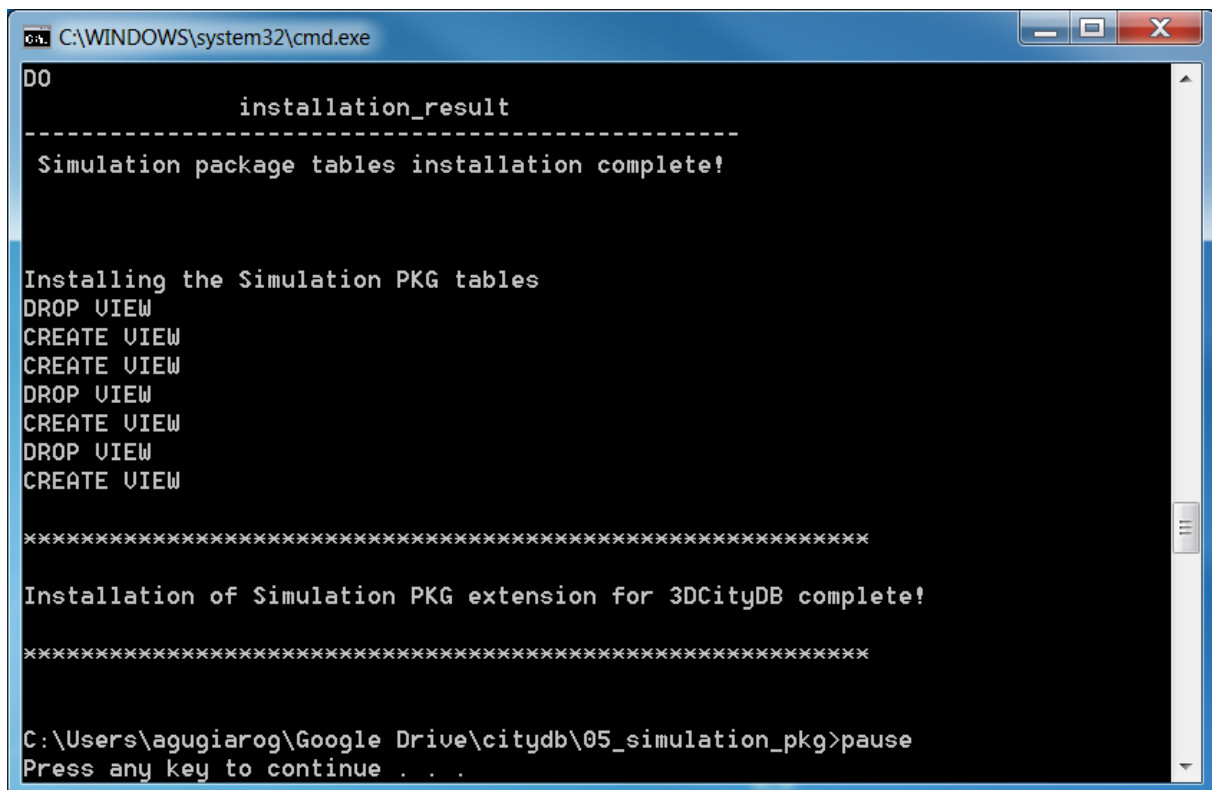
Upon successful installation, the message shown in Figure 4 will be output.



```

1 REM Shell script to install the Simulation PKG for the 3DCityDB
2 REM on PostgreSQL/PostGIS
3
4 REM Provide your database details here
5 set PGPORT=write here the port of the postgres server (e.g. 5432)
6 set PGHOST=write here the name/address of the host (e.g. localhost)
7 set PGUSER=write here the postgres user (e.g.) postgres
8 set CITYDB=write here the name of the database (e.g. vienna)
9 set PGBIN=write here the path to psql.exe (e.g. C:\Program Files\PostgreSQL\9.6\bin)
10
11 REM cd to path of the shell script
12 cd /d %~dp0
13
14 REM Run INSTALL_Simulation_PKG.sql to install the Simulation PKG into the 3DCityDB instance
15 "%PGBIN%\psql" -h %PGHOST% -p %PGPORT% -d "%CITYDB%" -U %PGUSER% -c "ALTER DATABASE %CITYDB% SE
16 "%PGBIN%\psql" -h %PGHOST% -p %PGPORT% -d "%CITYDB%" -U %PGUSER% -f "INSTALL_Simulation_PKG.sql
17 pause
  
```

Figure 3: Example of the CREATE\_DB\_Simulation\_PKG.bat batch file



```

D0
      installation_result
-----
Simulation package tables installation complete!

Installing the Simulation PKG tables
DROP VIEW
CREATE VIEW
CREATE VIEW
DROP VIEW
CREATE VIEW
DROP VIEW
CREATE VIEW

*****

Installation of Simulation PKG extension for 3DCityDB complete!

*****

C:\Users\agugiarog\Google Drive\citydb\05_simulation_pkg>pause
Press any key to continue . . .
  
```

Figure 4: Message upon successful installation

## 5.3 Manual installation

In order to install the 3DCityDB Simulation Package, the user needs to perform the steps described in the following.

### Step 1 – Identify the owner of the citydb schema

As seen before, the installation of the 3DCityDB extension should conveniently be carried out using the same user that installed the *citydb* schema and all included relations.

### Step 2 – Execute the SQL scripts in folder `\05_simulation_pkg\postgresql`

The SQL scripts in the folder `\05_simulation_pkg\postgresql`

01\_Simulation\_PKG\_FUNCTIONS.sql,  
02\_Simulation\_PKG\_DML\_FUNCTIONS.sql,  
03\_Simulation\_PKG\_TABLES.sql,  
04\_Simulation\_PKG\_VIEWS.sql

must simply run **sequentially** as seen before.

Upon successful installation, the message shown in Figure 5 will be output.

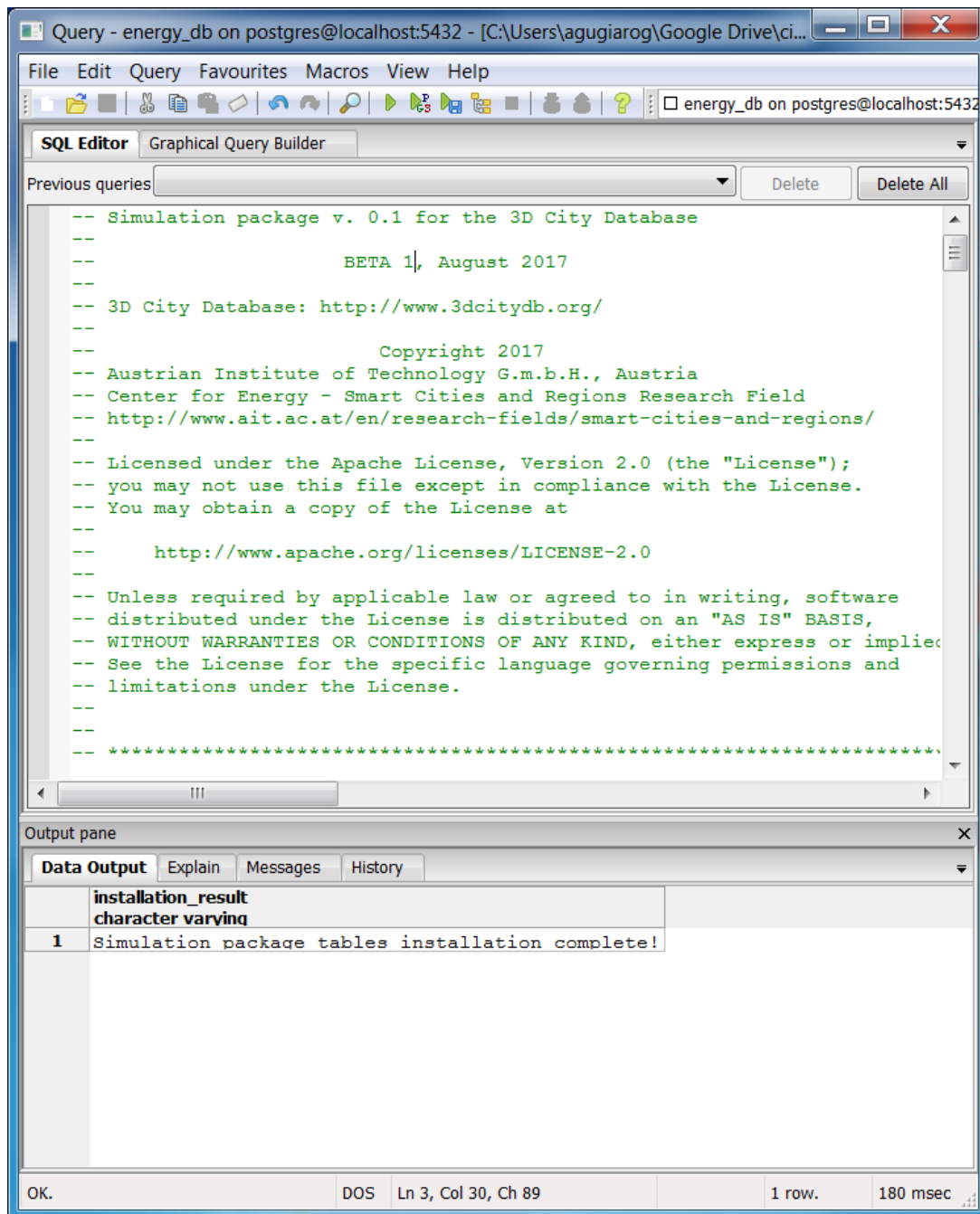


Figure 5: Message upon successful installation of the Simulation Package for the 3DCityDB

## 5.4 Installation of the test data

All test data are contained within on single SQL script file, named `Simulation_Pkg_ADE_TEST_DATA.sql` and located in the `\05_simulation_pkg\test_data` folder.

The installation process is the same as before: the script can be run from within an SQL command window of PgAdmin.



## 6 Document revision notes

Date	Author	Notes
15 September 2017	Agugiaro Giorgio	First draft release referring to Simulation Package 0.1 beta version
15 May 2018	Agugiaro Giorgio	Correction of typos, updated UML diagram, and ER diagram. Improved text. Added Chapter 6 “Document revision notes”





## Notes

## Notes