3D City Database extension

for the

CityGML Energy ADE 0.8

PostgreSQL Version

Documentation

Last update: 28 August 2017

AUSTRIAN INSTITUTE OF TECHNOLOGY
TOMORROW TODAY

Active participants in development

Name	Institution	Email
Giorgio Agugiaro Patrick Holcik	AIT – Austrian Institute of Technology G.m.b.H. Center for Energy – Smart Cities & Regions	giorgio.agugiaro@ait.ac.at p.holcik@gmail.com

Acknowledgements

The 3D City Database extension for the CityGML Energy ADE 0.8 (PostgreSQL version) was partially developed in the framework of the following international projects:

The European FP7-PEOPLE-2013 Marie Curie Initial Training Network "**CINERGY**" project (Grant Agreement Number 606851). Homepage: http://www.ci-nergy.eu	
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://integrcity.epfl.ch	

Finally, the authors would like to thank Thomas Kolbe and Zhihang Yao (TU München), Claus Nagel (virtualcitySYSTEMS) and Alexandru Nichersu (EIFER) for their suggestions and the fruitful discussions.

Table of Contents

Disclaimer

1 Introduction

The Energy Application Domain Extension (Energy ADE) extends the CityGML data model in order to add several energy-related entities and attributes which are particularly required in order to develop applications dealing with Urban Energy Planning.

With the Energy ADE reaching a good level of maturity and stability, and as already expressed by several members of the Energy ADE Development Group during the past workshops (e.g. in Vienna in May 2016), there is a growing interest and need in having a suitable database implementation of the Energy ADE, ideally extending the already available free and open-source 3D City Database (3DCityDB).

However, the current version of the 3DCityDB Import/Export tool doesn't provide generic solutions for handling CityGML ADE elements in CityGML instance documents. As a matter of fact, research work is already being carried out at the Technical University of Munich and by the 3DCityDB development team in order to extend to 3DCityDB Import/Export tool to deal with *any* ADE and to *automatically* perform the mapping between the object-oriented model and the relational-database model, in order to derive database schemas "on the fly". Further details can be read in Yao and Kolbe (2017)[1].

While waiting for this planned and powerful upgrade of the 3DCityDB Import/Export tool to be fully developed, tested, optimised, implemented and finally released to the public, a manually derived database schema is presented and provided together with this document. Initial work started internally at AIT during v. 0.6 of the Energy ADE (Agugiaro, 2016)[2], and it was then adapted to the current v. 0.8. As reported by the Energy ADE developers, similar and parallel (sometimes partial) implementations have been carried out in the past by different groups, generally tailored to specific project needs. However, a common, shared (and publicly available) implementation has not been made available as of now.

This 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 Energy ADE 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] Yao, Z., Kolbe, T.H., 2017, *Dynamically Extending Spatial Databases to support CityGML Application Domain Extensions using Graph Transformations*. 37. Wissenschaftlich-Technische Jahrestagung der DGPF in Würzburg – Publikationen der DGPF, Band 26.
http://www.dgpf.de/src/tagung/jt2017/proceedings/proceedings/papers/30_DGPF2017_Yao_Kolbe.pdf

[2] Agugiaro, G., 2016, *Enabling "energy-awareness" in the semantic 3D city model of Vienna*. ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., IV-4/W1, pp. 81-88, doi:10.5194/isprs-annals-IV-4-W1-81-201.
http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W1/81/2016/isprs-annals-IV-4-W1-81-2016.pdf

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 Energy ADE requires the latest version of the 3DCityDB (v. 3.3.0) 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) Define a non-concurrent way of extending the 3DCityDB with other ADEs (e.g. the Utility Networks ADE) at the same time, for example adopting naming prefixes to avoid potential homonymy conflicts with other ADEs or standard 3DCityDB objects;
b) Build upon the existing objects of the 3DCityDB (relations, stored procedures, etc.), but keep the original ones untouched in order to guarantee the normal usage of the Importer/Exporter tool with "plain" CityGML instance documents;
c) Try to stay as close as possible to the original design "style" of the 3DCityDB when it comes to tables, constraints, naming conventions, data types, etc.;
d) Follow the v. 0.8 of the Energy ADE 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 Energy ADE as indicated in the UML diagrams. In addition, in some, limited circumstances, little changes/improvements were made, mostly anticipating change requests for the v. 0.9. Such changes are described and explained in the text whenever they occur;
e) Keep the number of tables in check, i.e. grouping and merging multiple classes into one table – whenever reasonable –, although at the cost of some inefficiencies (e.g. in terms of disk space consumption) or some design "inelegances";
f) 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 Energy ADE documentation at hand. The **3DCityDB documentation** can accessed on-line at this URL:

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

while the **Energy ADE documentation** here:

https://github.com/cstb/citygml-energy/blob/master/doc/guidelines

Finally, especially for those users who are completely new to the 3DCityDB, 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

2 Data Modelling

The following pages cite several parts of the of CityGML specification and of the Energy ADE guidelines which are necessary for a better understanding. Design decisions in the model are explicitly visualised within the UML diagrams. Following parts of the Energy ADE are presented in detail:

- Time series and Schedule module,
- Material and Construction module,
- Building Physics module,
- Occupancy module,
- Energy Systems module

In addition, whenever necessary, parts of the CityGML UML diagrams are also depicted, in order to better show where the Energy ADE elements connect to the existing classes.

For intuitive understanding, classes which will be merged to a single table in the relational schema, are shown as orange blocks in the UML diagrams, while n:m relations, which only can be represented by additional tables, are represented as green blocks.

Please note that the UML diagrams presented in the following Figures have been simplified for better readability, i.e. diagrams regarding codelists and enumerations are omitted. Appendix B at the end of this document contains the complete representation of all UML diagrams of the Energy ADE v. 0.8.

2.1 Time series and Schedule module

This module introduces two classes: _TimeSeries and _Schedules, essential to model the time-depending inputs and results, e.g. urban energy analyses. These classes are used in other modules of the Energy ADE.

All _TimeSeries subclasses share some common properties, gathered in the TimeValuesProperties class, regarding for example the acquisition method, the type of interpolation, the source of the time series data, etc. Please refer to Figure 1 for the UML diagram.

Time series can be either regular or irregular. Regular time series contain values generated at regularly spaced interval of time, over a given temporal extent (i.e. start, end and duration time). In irregular time series, data follows a temporal sequence, but the measurement points may not happen at a regular time interval. Therefore, each value must be associated with its own timestamp.

Time series values may be stored in an external file (e.g. csv or text), both for regular (RegularTimeSeriesFile class) and irregular time series (IrregularTimeSeriesFile class). A number of attributes must be provided as metadata about the file, for example about the decimal symbol, the record and field separators, etc.

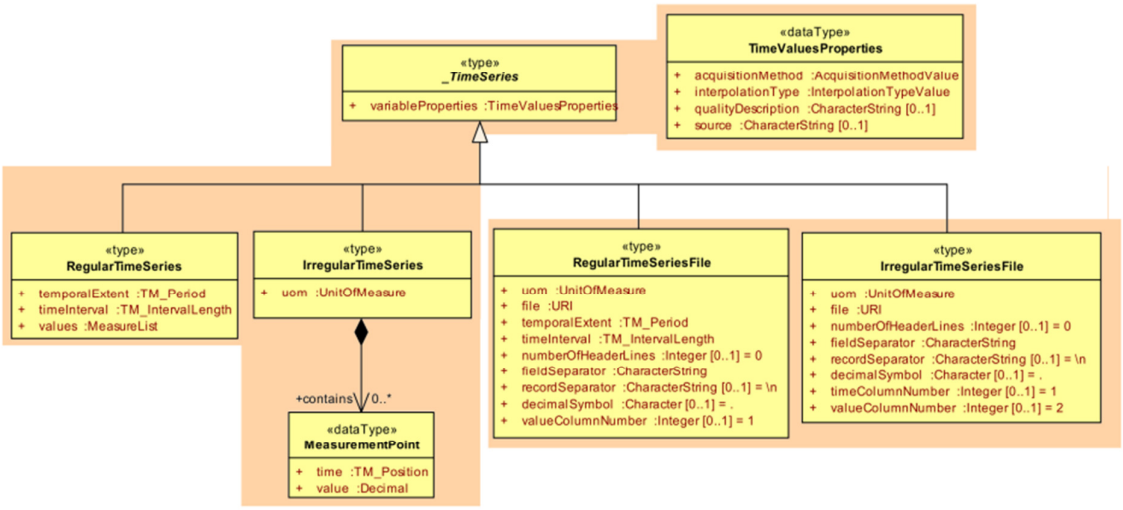


Figure 1: UML diagram of TimeSeries

Schedules can be modelled in four possible ways and are derived from the abstract class _Schedule, depending on the available information and the application requirements. They range from a simple constant value to a detailed schedule characterised by time depending values (i.e. a _TimeSeries object), with DualValue and DailyPattern schedules in between. DailyPattern schedules are meant to model typical days (and periods of year) which are not specific to a specific year. Please refer to Figure 2 for the UML diagram.

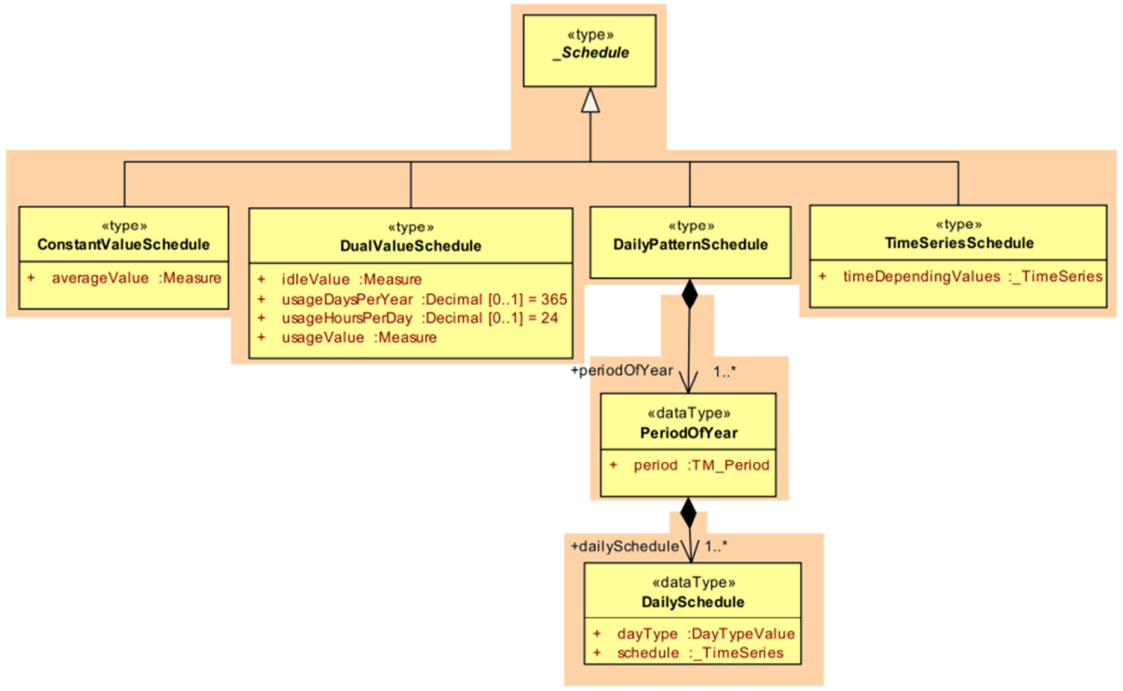


Figure 2: UML diagram of Schedules

2.2 Material and Construction module

The Material and Construction module characterises the building construction parts, detailing their structure and specifying their thermal and optical properties. Please refer to Figure 3 for the UML diagram.

The central class is AbstractConstruction, which is specialised into a Construction and a ReverseContruction class. A Construction consists of an (ordered) set of layers, each one made of one (or more) layer components. Each layer component is made of one (or more) materials, which can be either a gas or a solid material. A ServiceOfLife class is used to characterise both Construction and LayerComponent classes. Finally, each construction is characterised also by a number of optical properties, such as transmittance, reflectance, etc. A ReverseConstruction class represents a construction (hence the association with Contruction by means of attribute baseConstruction), however with an inverted order of layers.

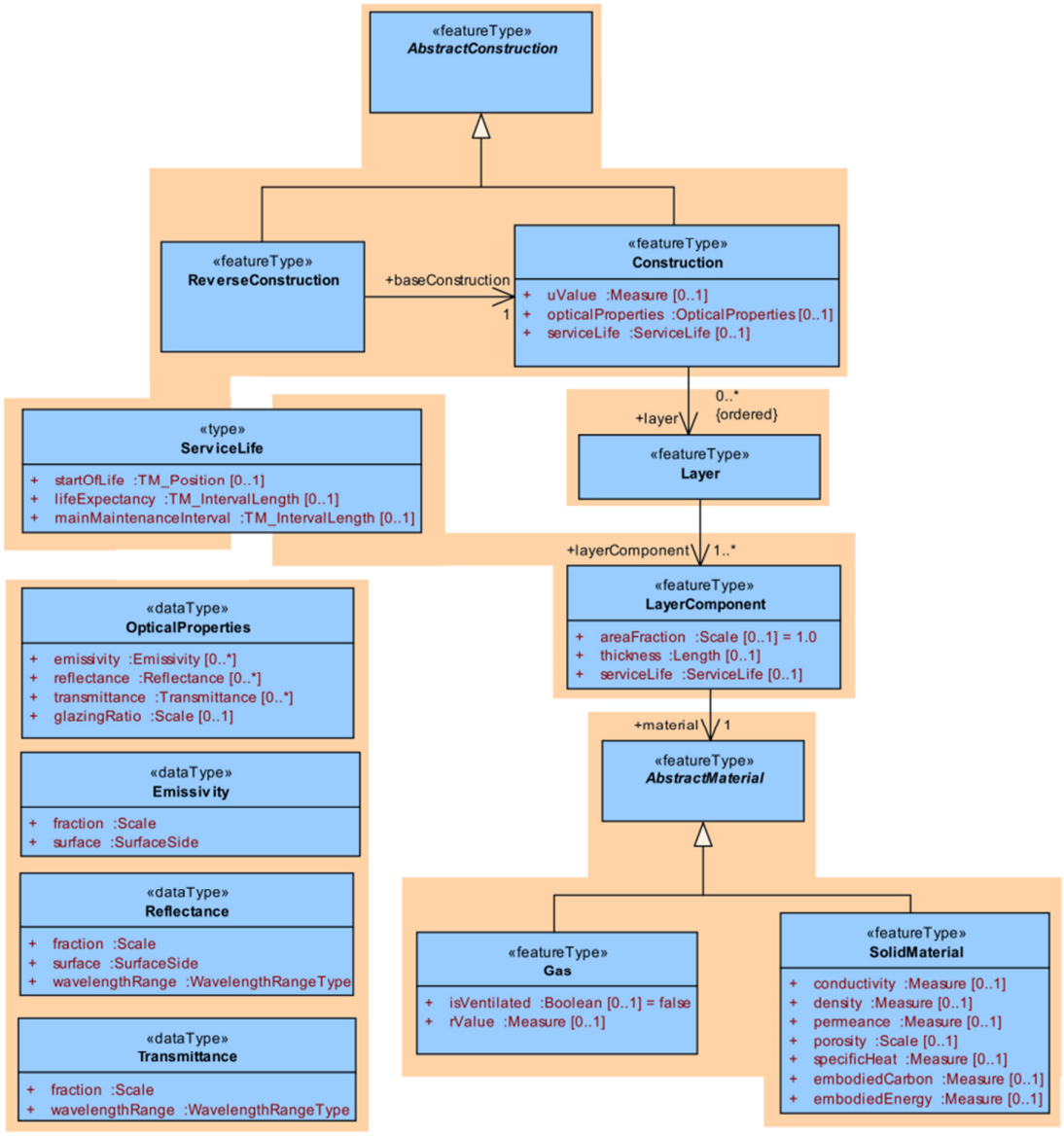


Figure 3: UML diagram of the Material and Construction module

2.3 Building Physics module

The Building Physics module extends the existing CityGML AbstractBuilding class, and relates it to new entities like ThermalZone, ThermalBoundary, and ThermalOpening. Please refer to Figure 4 for the UML diagram.

The “central” object is the ThermalZone, which is the reference volume for heating/cooling energy demand calculation. An AbstractBuilding can have multiple thermal zones, and each of them is bounded by a number of thermal boundaries. Thermal boundaries can have thermal openings. Two adjacent thermal zones are separated by a shared thermal boundary.

Each ThermalZone, ThermalBoundary and ThermalOpening class is derived from a _CityObject class. Each of them is characterized by a number of attributes. For some of them, multiple values are possible, leading to a cardinality of 0..*, e.g. like in the case of volume, floor area, energy performance certification, and refurbishment measures.

Every _CityObject class can have one or multiple WeatherData objects. These represent time series of measured or processed meteorological or radiation parameters, defined by three properties: the type of the weather data, the time series of its values, and optionally the position of the sensor.

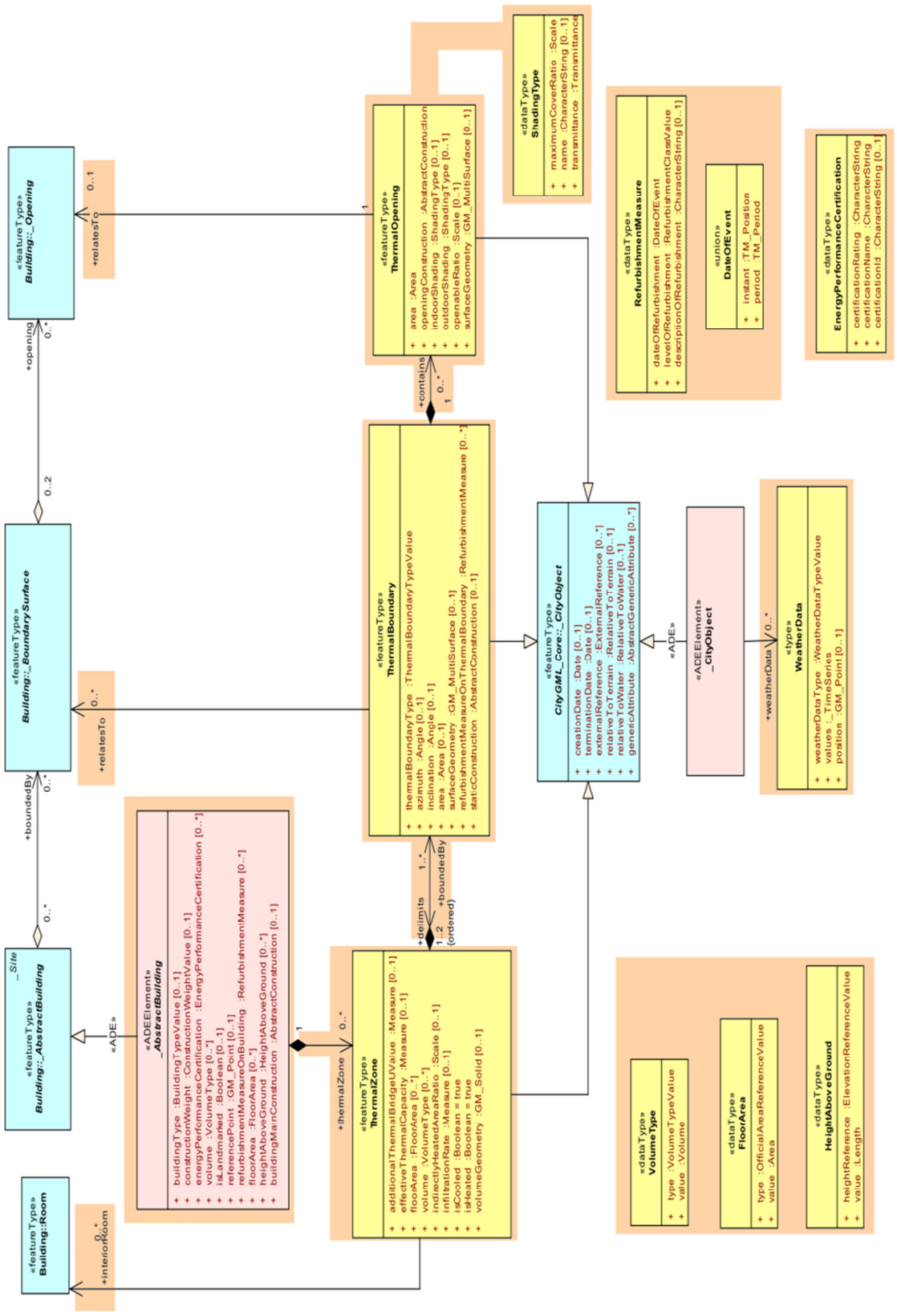


Figure 4: UML diagram of the Building Physics module

2.4 Occupancy module

The Occupancy module contains the detailed characterisation of the building usage, i.e. people and facilities. Please refer to Figure 5 for the UML diagram.

The main classes are the UsageZone and the BuildingUnit. A usage zone is contained in a thermal zone and characterises the usage of an homogeneous part of the building in terms of heating, cooling, ventilation etc. A building unit is a part of a single usage zone which can be defined as a subdivision of a building with its own lockable access from the outside or from a common area. A building unit is related to one or more occupants, such as a dwelling or a workplace.

The Occupants class identifies a homogeneous group of occupants of a usage zone or building unit: occupants are characterized by a given number of persons which occupy the corresponding zone following a certain time schedule. For zone thermal modelling purposes, the heat dissipation of this occupant group is also specified. For residential occupants, the Occupants class may contain one or more Household objects.

The Facilities class represents devices which consume and dissipate energy. Each UsageZone or BuildingUnit class can have one or more _Facilities.

Both the UsageZone and the BuildingUnit classes are derived from class _CityObject.

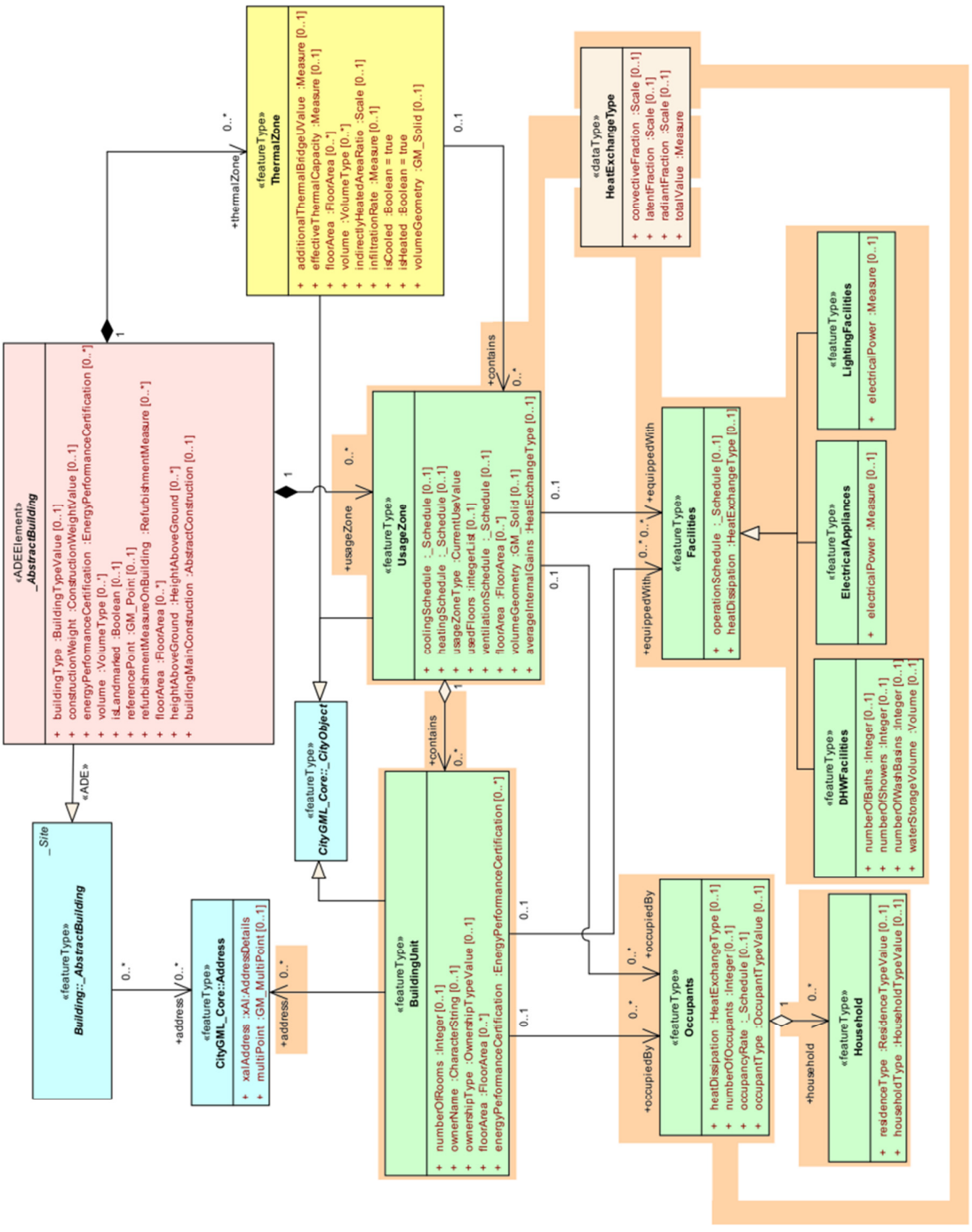


Figure 5: UML diagram of the Occupancy module

2.5 Energy Systems module

The Energy Systems module contains the energy forms (energy demand and sources) and energy systems (conversion, distribution and storage systems) to perform energy demand and supply analyses. Please refer to Figure 3 for the UML diagram. Please refer to Figure 6 and Figure 7 for the UML diagram. Given the size of the UML diagram, the Energy Systems module is split:. The *trait d'union* between the two Figures is the EnergyConversionSystem class.

The main classes of this module are the EnergyDemand, the EnergyConversionSystem, the EnergyStorageSystem and the EnergyDistributionSystems and their respective subclasses.

The EnergyDemand represents the energy required to satisfy a specific end-use (e.g. space heating, space cooling, domestic hot water) of a given _CityObject. Such energy is stored or distributed by means of thermal or power systems.

In order to satisfy the energy demand, an energy conversion system is required which, depending on certain system operation settings (class SystemOperation), either produces or consumes energy (class FinalEnergy). There are a number of specific EnergyConversionSystem subclasses, e.g. in the case of the _SolarEnergySystem ones, which are needed to further derive and characterise solar thermal or photovoltaic panels.

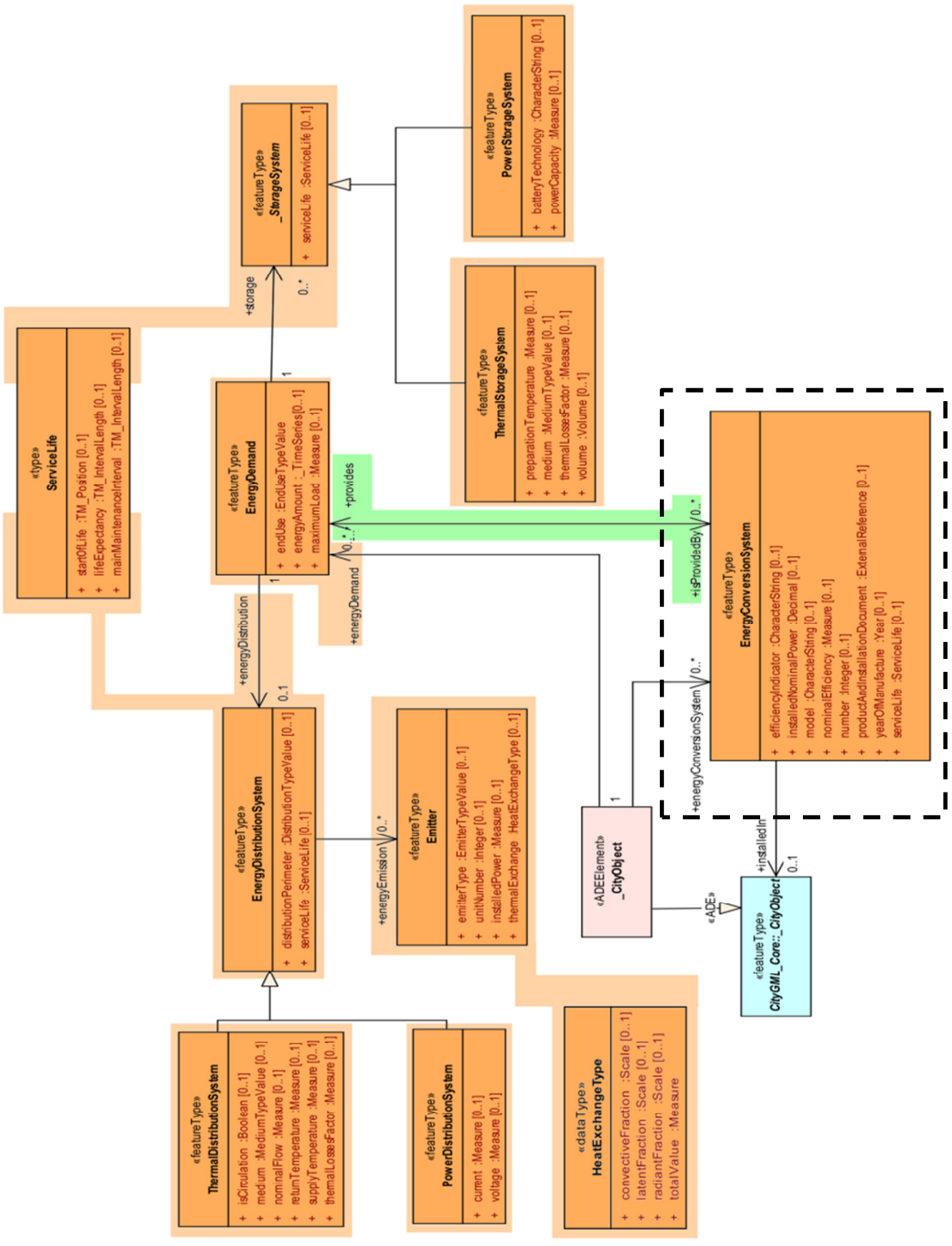


Figure 6: UML diagram of the Energy Systems module (part A). The EnergyConversionSystem class (inside the dashed box) acts as connection between this schema and the one in Figure 7

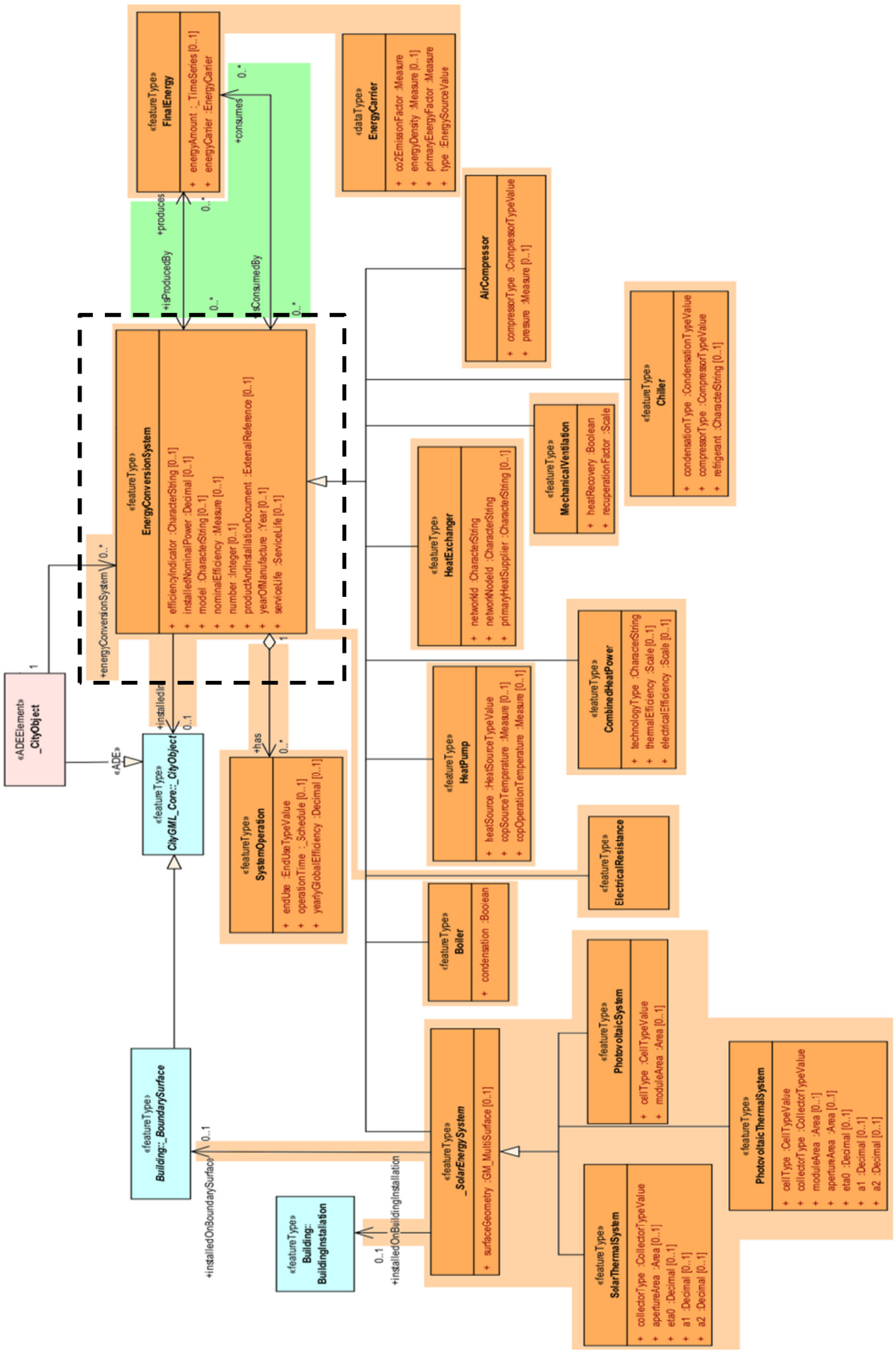


Figure 7: UML diagram of the Energy Systems module (part B). The EnergyConversionSystem class (inside the dashed box) acts as connection between this schema and the one in Figure 6

3 Database design

This section presents and describes the implementation of the Energy ADE as a relational database schema for PostgreSQL. However, before proceeding with the Energy ADE itself, a number of changes and additions to the current "vanilla" 3DCityDB implementation are required in order to add support to Application Domain Extensions.

3.1 ADE support in 3DCityDB

As mentioned in the system design decisions (section 1.1), the 3DCityDB should allow for support of different ADEs at the same time. Moreover, it should be possible to store metadata information about *any* ADE (e.g. name, version, xml schema location, etc.), avoid potential homonymy conflicts among tables belonging to different ADEs or standard 3DCityDB objects, and to provide a flexible way of defining/extending stored procedures to facilitate the overall database management.

For these reasons, a set of rules were first defined and then implemented, in order to prepare the 3DCityDB to be extended by an Application Domain Extension. Their implementation can be roughly summarised in three main points, which will be described in the following.

a) Add a Metadata module to "register" ADEs;
b) Define rules to map ADE classes to new or existing tables;
c) Define rules to deal with database stored procedures.

Please note that all rules were agreed upon within the 3DCityDB development team and are being gradually tested and implemented for the next official 3DCityDB release. Besides, they are part of a major development plan which will add dynamic ADE support to other relevant tools for CityGML (e.g. Importer/Exporter, Spreadsheet generator, etc.). However, due to the still on-going development cycle, the implementation described in this document might eventually not coincide with the final one. Nevertheless, particular attention will be paid to keep the development of the Energy ADE database extension in line and compatible with that of the 3DCityDB.

3.1.1 Metadata module

The Metadata module is meant to allow for "registration" of an ADE upon installation, in that all relevant information and metadata are stored in the 3DCityDB. It obeys to the rule that every ADE must be self-consistent, i.e. references to CityGML schemas are allowed, but not to *other* ADEs. As long as this constraint is respected, an ADE itself can be composed of several schemas.

What is more, in order to avoid homonymy conflicts among tables belonging to different ADEs or "vanilla" CityGML, every ADE must be assigned a *unique* **db_prefix**, which will be used consistently in the 3DCityDB to identify uniquely all entities (tables, stored procedures, triggers, etc.) derived from that ADE. For example, in the case of the Energy ADE 0.8, the db_prefix "nrg8" is used, while for the future Energy ADE v. 0.9 the db_prefix "nrg9" could be adopted. For others ADEs, the same principle still holds, as exemplified in Figure 9 (see

later about the **ADE** table). In this document, the simple "nrg" db_prefix will be used in the following text for better readability. Therefore, the names of all tables, functions, views, etc. will be prefixed with "nrg" + "_" (underscore). Please note that the actual db_prefix used in the figures of this document and in the database scripts of the 3DCityDB extension for Energy ADE might vary according to the current release version.

The Metadata module consists of a number of new tables that are added to the *citydb* schema, while the **OBJECTCLASS** table is extended, as shown in Figure 8. In the following, the three most important tables will be described, namely the **ADE**, the **SCHEMA** and the **OBJECTCLASS** tables.

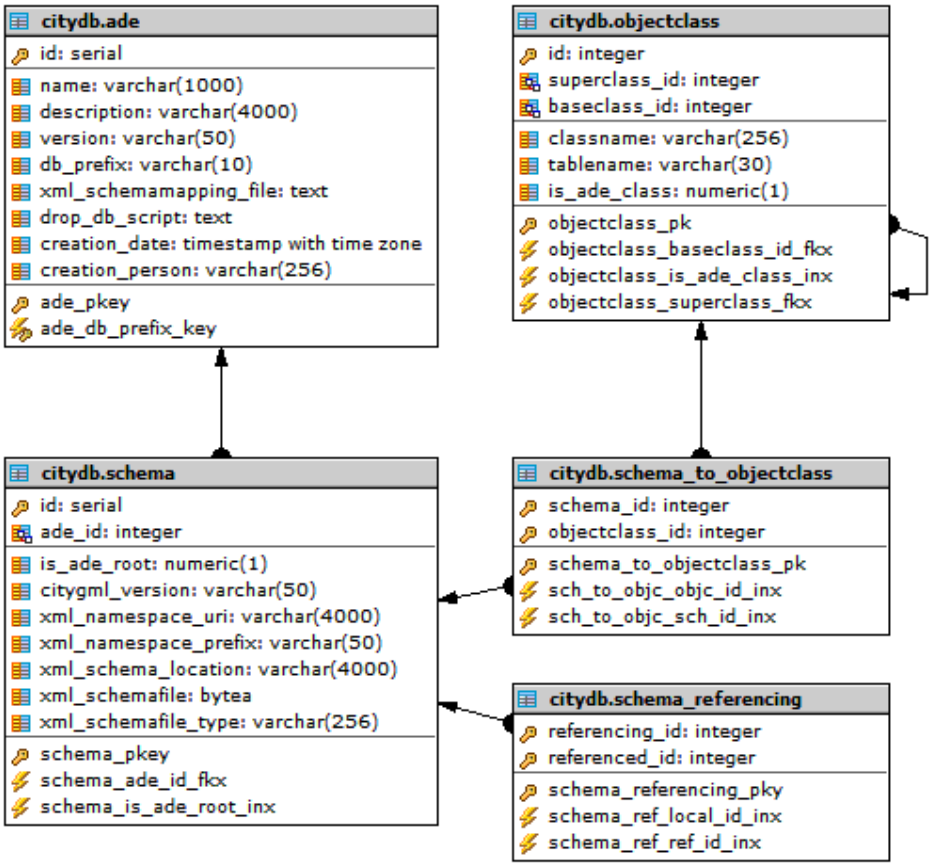
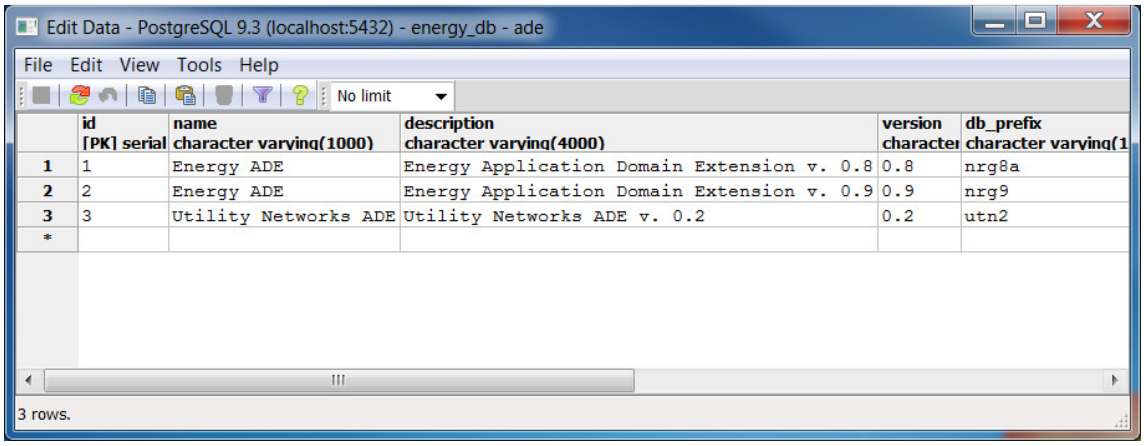


Figure 8: ER model of the Metadata module

In table **ADE**, the basic information of each ADE is stored. The name, the description, the version and the db_prefix are stored, among the rest. Please note that the DB_PREFIX column must contain *unique not null* values.



	id [PK] serial	name character varying(1000)	description character varying(4000)	version character	db_prefix character varying(1
1	1	Energy ADE	Energy Application Domain Extension v. 0.8	0.8	nrg8a
2	2	Energy ADE	Energy Application Domain Extension v. 0.9	0.9	nrg9
3	3	Utility Networks ADE	Utility Networks ADE v. 0.2	0.2	utn2
*					

Figure 9: Example of data contained in the ADE table (excerpt)

In table **SCHEMA**, all relevant information about the schema(s) of an ADE, as well as those of CityGML 1.0 and 2.0, are stored. Please refer to Figure 27 in the Installation chapter for an example.

Regarding the **OBJECTCLASS** table, this is the table where all class names (column CLASSNAME) of the CityGML schema are generally managed. The relation of the subclass to its parent class is represented via the column SUPERCLASS_ID in the subclass as a foreign key to the ID of the parent class. For ADE support, the OBJECTCLASS table is extended by a number of columns, namely the IS_ADE_CLASS, BASECLASS_ID and the TABLENAME ones.

The IS_ADE_CLASS column allows to discriminate between "vanilla" CityGML classes (value is 0) and ADE classes (value is 1). The BASECLASS_ID column contains a foreign key to the ID of the uppermost class (i.e. the base class) of each class, while the TABLENAME column contains the name of the table where (most of) the data of an object are stored. The TABLENAME column is constrained to contain values up to 30 characters of length. This implements one of the database design rules according to which no table names can be longer than 30 character (including the prefix). This is meant to guarantee compatibility with other database systems (e.g. Oracle) in case of code porting.

Upon installation of the Metadata module, the "vanilla" CityGML object classes already contained in the table are also updated, as shown in Figure 10.

Regarding all new ADE classes, these have to be inserted into the OBJECTCLASS table upon installation of a new ADE. The first available ID is assigned to the first new class, etc. An example for the Energy ADE is given in Figure 11.

Please note: in the current implementation, the Energy ADE classes are assigned hard-coded ID values (200 till 263), *de facto* leaving a gap between the last "vanilla" CityGML class (_WaterObject, ID=105). This is, however, a temporary implementation decision in order to facilitate testing and debugging. All remaining database objects (stored procedures, views, etc.) are already implemented to deal with dynamic ID assignment of the object classes.

Further details on the use and function of the OBJECTCLASS table can be found in the 3DCityDB documentation.

Edit Data - PostgreSQL 9.3 (localhost:5432) - energy_db - objectclass

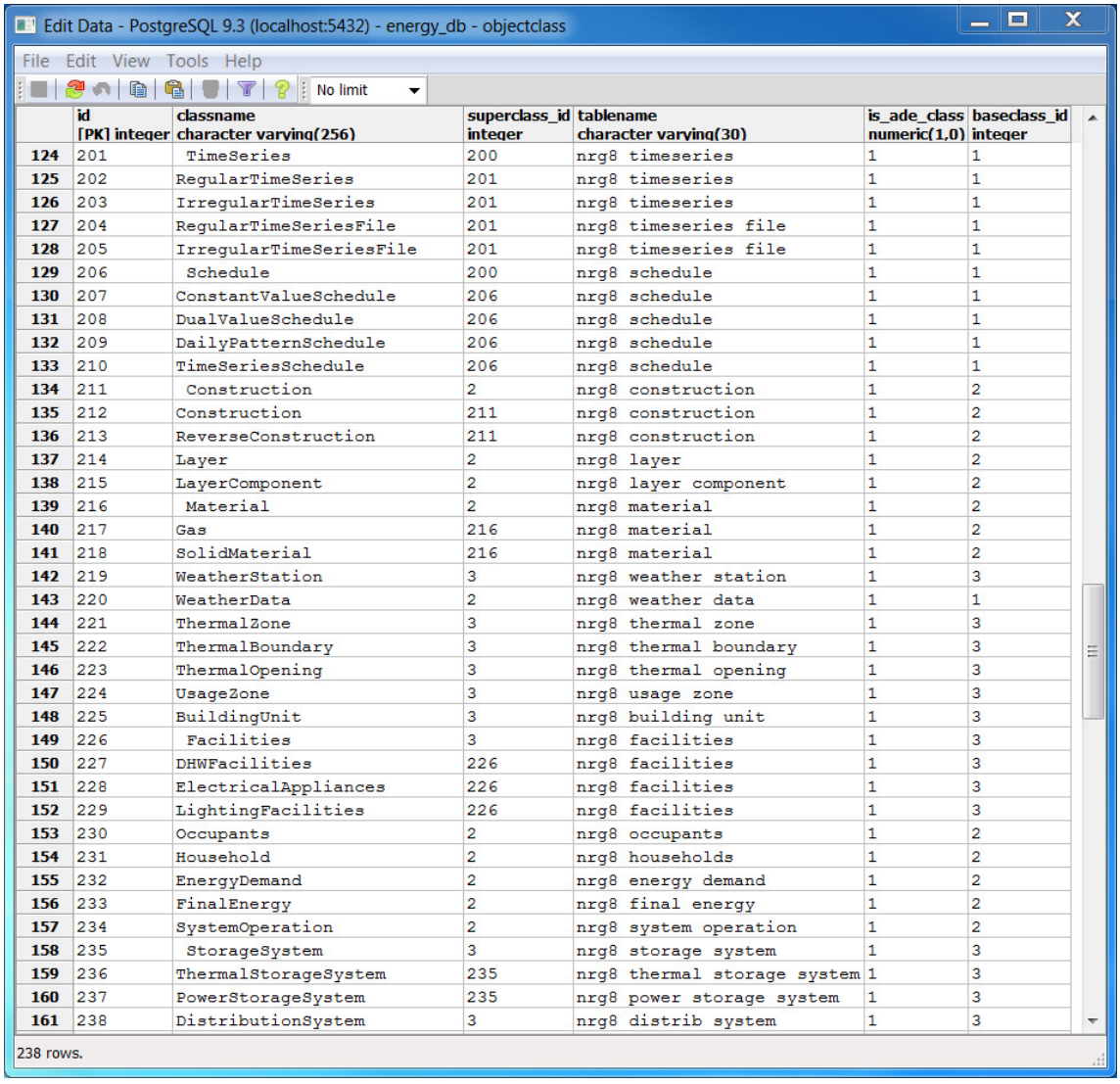
File Edit View Tools Help

No limit

	id [PK] integer	classname character varying(256)	superclass_id integer	tablename character varying(30)	is_ade_class numeric(1,0)	baseclass_id integer
1	0	Undefined			0	
2	1	GML		cityobject	0	
3	2	Feature	1	cityobject	0	1
4	3	CityObject	2	cityobject	0	2
5	4	LandUse	3	land use	0	3
6	5	GenericCityObject	3	generic cityobject	0	3
7	6	VegetationObject	3	cityobject	0	3
8	7	SolitaryVegetationObject	6	solitary vegetat object	0	3
9	8	PlantCover	6	plant cover	0	3
10	9	WaterBody	105	waterbody	0	3
11	10	WaterBoundarySurface	3	waterboundary surface	0	3
12	11	WaterSurface	10	waterboundary surface	0	3
13	12	WaterGroundSurface	10	waterboundary surface	0	3
14	13	WaterClosureSurface	10	waterboundary surface	0	3
15	14	ReliefFeature	3	relief feature	0	3
16	15	ReliefComponent	3	relief component	0	3
17	16	TINRelief	15	tin relief	0	3
18	17	MassPointRelief	15	masspoint relief	0	3
19	18	BreaklineRelief	15	breakline relief	0	3
20	19	RasterRelief	15	raster relief	0	3
21	20	Site	3	cityobject	0	3
22	21	CityFurniture	3	city furniture	0	3
23	22	TransportationObject	3	cityobject	0	3
24	23	CityObjectGroup	3	cityobjectgroup	0	3
25	24	AbstractBuilding	20	building	0	3
26	25	BuildingPart	24	building	0	3
27	26	Building	24	building	0	3
28	27	BuildingInstallation	3	building installation	0	3
29	28	IntBuildingInstallation	3	building installation	0	3
30	29	BuildingBoundarySurface	3	thematic surface	0	3
31	30	BuildingCeilingSurface	29	thematic surface	0	3
32	31	InteriorBuildingWallSurface	29	thematic surface	0	3
33	32	BuildingFloorSurface	29	thematic surface	0	3
34	33	BuildingRoofSurface	29	thematic surface	0	3
35	34	BuildingWallSurface	29	thematic surface	0	3
36	35	BuildingGroundSurface	29	thematic surface	0	3
37	36	BuildingClosureSurface	29	thematic surface	0	3

163 rows.

Figure 10: Excerpt of the OBJECTCLASS table after installation of the Metadata module. The columns TABLENAME, IS_ADE_CLASS and BASECLASS_ID are added



	id [PK] integer	classname character varying(256)	superclass_id integer	tablename character varying(30)	is_ade_class numeric(1,0)	baseclass_id integer
124	201	TimeSeries	200	nrg8 timeseries	1	1
125	202	RegularTimeSeries	201	nrg8 timeseries	1	1
126	203	IrregularTimeSeries	201	nrg8 timeseries	1	1
127	204	RegularTimeSeriesFile	201	nrg8 timeseries file	1	1
128	205	IrregularTimeSeriesFile	201	nrg8 timeseries file	1	1
129	206	Schedule	200	nrg8 schedule	1	1
130	207	ConstantValueSchedule	206	nrg8 schedule	1	1
131	208	DualValueSchedule	206	nrg8 schedule	1	1
132	209	DailyPatternSchedule	206	nrg8 schedule	1	1
133	210	TimeSeriesSchedule	206	nrg8 schedule	1	1
134	211	Construction	2	nrg8 construction	1	2
135	212	Construction	211	nrg8 construction	1	2
136	213	ReverseConstruction	211	nrg8 construction	1	2
137	214	Layer	2	nrg8 layer	1	2
138	215	LayerComponent	2	nrg8 layer component	1	2
139	216	Material	2	nrg8 material	1	2
140	217	Gas	216	nrg8 material	1	2
141	218	SolidMaterial	216	nrg8 material	1	2
142	219	WeatherStation	3	nrg8 weather station	1	3
143	220	WeatherData	2	nrg8 weather data	1	1
144	221	ThermalZone	3	nrg8 thermal zone	1	3
145	222	ThermalBoundary	3	nrg8 thermal boundary	1	3
146	223	ThermalOpening	3	nrg8 thermal opening	1	3
147	224	UsageZone	3	nrg8 usage zone	1	3
148	225	BuildingUnit	3	nrg8 building unit	1	3
149	226	Facilities	3	nrg8 facilities	1	3
150	227	DHWFacilities	226	nrg8 facilities	1	3
151	228	ElectricalAppliances	226	nrg8 facilities	1	3
152	229	LightingFacilities	226	nrg8 facilities	1	3
153	230	Occupants	2	nrg8 occupants	1	2
154	231	Household	2	nrg8 households	1	2
155	232	EnergyDemand	2	nrg8 energy demand	1	2
156	233	FinalEnergy	2	nrg8 final energy	1	2
157	234	SystemOperation	2	nrg8 system operation	1	2
158	235	StorageSystem	3	nrg8 storage system	1	3
159	236	ThermalStorageSystem	235	nrg8 thermal storage system	1	3
160	237	PowerStorageSystem	235	nrg8 power storage system	1	3
161	238	DistributionSystem	3	nrg8 distrib system	1	3

238 rows.

Figure 11: Excerpt of the OBJECTCLASS table after installation of the 3DCityDB extension for the Energy ADE. The ADE classes are added to the OBJECTCLASS table according to the same rules for the "vanilla" classes. The only remarkable difference is the value in the IS_ADE_CLASS column, here set to 1 (i.e. TRUE)

3.1.2 Mapping rules for ADE classes

In general, but also following the rules established for the 3DCityDB, one or more classes of the UML diagrams are mapped onto one table. The name of the table is identical to the class name in most of the cases (a leading underscore indicating an abstract class is left out). Every ADE table is prefixed appropriately as described in the previous sections (e.g. "nrg_").

Classes may be combined into a single table according to the class relations as shown in the UML diagrams in section 2 (Data Modelling) by using orange-coloured boxes. The scalar attributes of the classes become columns of the corresponding table with identical (or very similar) names. For every attribute including measure information an additional _UNIT-suffixed column is provided to specify the unit of measurement. Please note that in the following sections not all attributes of every single table will be explained in detail. For those whose name and type can be easily recognised from the ULM diagrams, no explicit

explanation will be given. As usual, the reader is invited to have the UML diagrams at hand for reference while reading. In any case, most of the attributes are commented in the SQL installation script by means of the "`COMMENT ON`…" PostgreSQL functionality.

The types of the attributes are customised to corresponding PostgreSQL data types (see Explicit declaration of class affiliation in Table 1). Some attributes of type date were mapped to TIMESTAMP WITH TIME ZONE to allow a more accurate storage of time values. The following table shows the data type mapping.

Data type mapping (excerpt)	
UML	**PostgreSQL / PostGIS**
String, anyURI	VARCHAR, TEXT
Integer	INTEGER
Double, gml:LengthType, etc.	NUMERIC
Boolean	NUMERIC(1)
Date	DATE, TIMESTAMP WITH TIME ZONE
Primitive Type (Color, TransformationMatrix, CodeType etc.)	VARCHAR
Enumeration	VARCHAR
GML Geometry	GEOMETRY

Table 1: Data type mapping. Source: 3DCityDB documentation

As already mentioned, the overall logic of the current 3DCityDB implementation should be followed/reflected as much as possible for design and usability coherence. Furthermore, as a general rule, *no changes to "vanilla" 3DCityDB tables are allowed*, in order to secure that the Importer/Exporter will continue working as usual, though no ADE data will be "seen". Whenever a new column should be added to an existing "vanilla" table, appropriate workarounds must be implemented. Keeping in mind these general design criteria, a number of rules to map ADE classes to tables were identified and implemented. They can be grouped into four main groups and are described in the following sections.

3.1.2.1 Mapping of ADE non-_CityObject classes

In general, classes which are not specifically derived from the _CityObject one (e.g. _Features) are mapped to own tables, similarly to 3DCityDB tables **ADDRESS, APPEARANCE,** or **EXTERNAL_REFERENCE**. Each table has its own primary key defined by a sequence. In addition a column OBJECTCLASS_ID containing a foreign key to table **OBJECTCLASS** must be added whenever storing data of an object class listed there.

In the implementation of the Energy ADE 0.8, this mapping rules were used for example for time series, schedules, constructions, materials, occupants, households, etc.

3.1.2.2 Mapping of ADE _CityObject classes

In case of ADE classes derived from _CityObject, they are mapped to database tables according to the same criteria of the "vanilla" 3DCityDB:

- Attributes inherited from class _CityObject are stored in table CITYOBJECT;

- The remaining attributes are stored in a new (prefixed) table linked to table CITYOBJECT (using column ID as foreign key). For example, attributes of class ThermalZone are mapped to tables CITYOBJECT *and* NRG_THERMAL_ZONE;
- Further ancillary existing 3DCityDB tables, when necessary, have to be used as well. For example, the surface geometries of classes ThermalBoundary and ThermalOpening have to be stored in the already existing SURFACE_GEOMETRY table using the rules defined and described in the 3DCityDB documentation;
- All new linked tables (e.g. NRG_THERMAL_ZONE) must include a column OBJECTCLASS_ID containing a foreign key to table **OBJECTCLASS**.

In the implementation of the Energy ADE 0.8, these mapping rules were used for example for thermal zones, thermal boundaries, thermal openings, usage zones, etc.

3.1.2.3 Mapping of ADE-extended classes

Similar to the previous case, new attributes of existing classes extended by an ADE are mapped to new (prefixed) tables which are linked to the table corresponding to the class they are extending. For example, the Energy ADE _AbstractBuilding class is mapped to table NRG_BUILDING, which is linked to table BUILDING. In case of attributes requiring ancillary tables, the same rules apply as before.

Please note that, coherently with the UML diagram, table NRG_THERMAL_ZONE has a column BUILDING_ID containing a foreign key to table NRG_BUILDING (and not to BUILDING!).

3.1.2.4 Mapping of relations between ADE and "vanilla" classes

Whenever relations between an ADE and a "vanilla" class exist, they should be modelled in a way that guarantees no changes to the "vanilla" table. For example, in case of 1:n and m:n relations between ADE and "vanilla" classes, this can be achieved by means of an association table between to two classes.

In the Energy ADE, for example, this is the case of the relations between classes ThermalOpening and _Opening, ThermalBoundary to _BoundarySurface, or BuildingUnit to Address.

3.1.3 Database stored procedures

Besides creating new tables, a new ADE has also to provide a number of stored procedures to help data management of the database, given the complexity of the schema. In the "vanilla" 3DCityDB", stored procedures are stored by default in the *citydb_pkg* schema and can be grouped approximately in the "delete", "get_envelope" and "miscellaneous" families. In particular, the "delete" stored procedures provide a convenient way of deleting objects having data spread over different tables, while the "get_envelope" stored procedures compute the 3D envelope of any object possessing geometries. Further details on the stored procedures implemented in the 3DCityDB can be found in the 3DCityDB documentation.

For all ADE stored procedures, similar rules apply: they must be saved in the *citydb_pkg* schema, they must obey to the naming criteria described before using the db_prefix, e.g.

`citydb_pkg.nrg_delete_thermal_zone(…)` or `citydb_pkg.nrg_get_envelope_thermal_zone(…)` for stored procedures which delete or compute the 3D envelope of thermal zone object, respectively. Please note that, here and in the remainder of the documentation, the arguments of the stored procedures are not shown and substituted with a "…" for better readability.

In general, the ADE stored procedures follow the structure and design of the "vanilla" ones, however some specific rules apply, as described in the following.

3.1.3.1 Delete stored procedures for "stand alone" ADE classes

This is the most common group of ADE classes for which delete stored procedures are developed. The implementation is analogous to that of the "vanilla" ones. For example, the associated data contained in all ancillary tables have to be deleted *before* the actual object in the corresponding table is deleted.

3.1.3.2 Delete stored procedures for ADE-extended classes

In case of ADE-extended classes, the new ADE attributes are stored in a new linked table. In this case, the ADE delete function takes care of deleting *only* the new ADE attributes, but not the whole object.

For example, the `citydb_pkg.nrg_delete_building(…)` stored procedure deletes only data in the NRG_BUILDING table (and the dependent objects like thermal zone, usage zones, etc., if necessary) but *not* the data in the CITYOBJECT or BUILDING tables.

In order to delete the data of the whole object, the corresponding "vanilla" stored procedure must be used. For example, `citydb_pkg.delete_building(…)` must take care of deleting all "vanilla" *and* any other ADE data. For such cases, an ADE-hook mechanism is added to the "vanilla" stored procedure (see later).

3.1.3.3 Get_envelope stored procedures

In general, the get envelope stored procedures work exactly like the "vanilla" ones and must be written for all those new ADE _CityObject-derived classes that are not extending existing "vanilla" ones. The get_envelope stored procedures return the 3D bounding box of the corresponding objects.

3.1.3.4 ADE-hook mechanism for "vanilla" stored procedures

In order to guarantee that particular stored procedures work also with ADEs (e.g. `citydb_pkg.delete_building(…))`, a so-called ADE-hook mechanism was implemented, in particular when it comes to:

- Delete stored procedures for ADE-extended classes: in this case, the "vanilla" `citydb_pkg.intern_delete_cityobject(…)` stored procedure is modified and extended;
- The higher level `citydb_pkg.delete_cityobject(…)` and `citydb_pkg.get_envelope_cityobject(…)` stored procedures are modified

and extended, in order to work with *any* CityObject. Besides, this guarantees also that stored procedures like `citydb_pkg.delete_cityobjectgroup(…)` keep working also with ADEs.

- Two more "vanilla" stored procedures are similarly modified and extended, namely: `citydb_pkg.objectclass_id_to_table_name(…)` and `citydb_pkg.cleanup_schema(…)`.

Upon installation of the Metadata module, the "vanilla" versions of these stored procedures are backed up before being updated. For more details, please refer to section 5.2 (Automatic full installation).

3.1.4 Comments

Finally, a general recommendation which applies both to the "Metadata module" and to any ADE is to provide hints and description of the database objects by means of comments using the "`COMMENT ON`…" PostgreSQL functionality. This can be useful in case the full name was shortened, or to give hints about the range of values (e.g. in case of fractions or efficiency values). Moreover, comments can be read directly from the database (e.g. with PgAdmin, see Figure 12) or automatically imported by other programs (e.g. when creating a PHP-based GUI).

Please note that, as this functionality might not be supported by other database software (e.g. Oracle), a workaround might be necessary in future when porting the code.

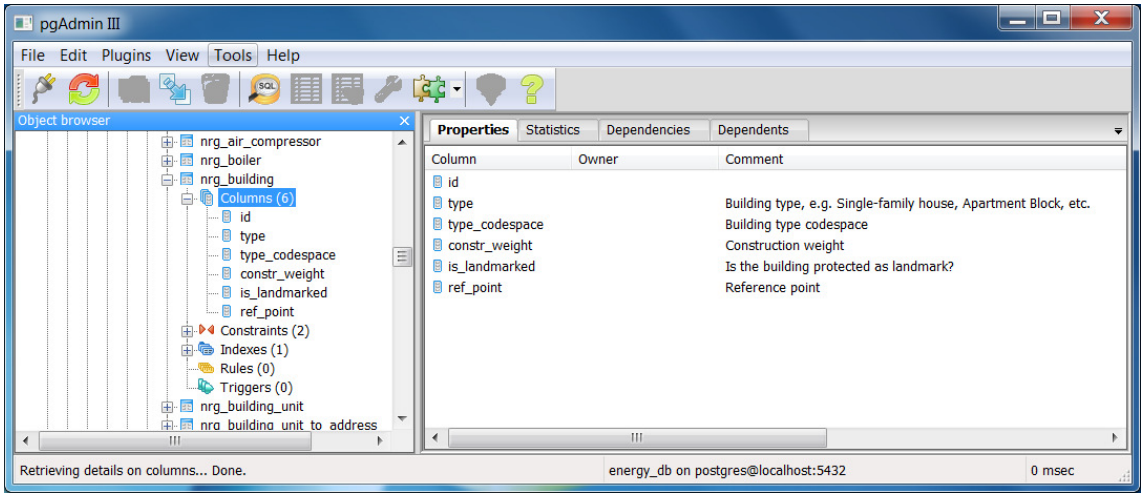


Figure 12: Example of comments for attributes

3.2 Database schema of the Energy ADE

In the following paragraphs, the tables of the relational schema are described in detail and displayed graphically. The description is based on the remarks on UML charts in section 2. Focus is put on situations where the conversion into tables leads to changes in the ER model.

In general, the database relations correspond to the Energy ADE v. 0.8. In a limited number of cases some modifications have been carried out on the base of expected changes which will mostly happen as soon as v. 0.9 is released. Such **implementation differences are highlighted in red** and explained whenever needed.

The figures are taken from PostgreSQL Maestro, a GUI-based admin tool for database management which offers a set of tools to edit and execute SQL scripts, build visual diagrams, etc. In future version of this document, the open-source tool pgModeler will be used to maintain the schema.

3.2.1 Time series and schedule module

For the _TimeSeries class (and derived/dependent classes), only two tables are created. Please refer to Figure 13 for the representation of the corresponding ER diagram.

In the **NRG_TIMESERIES** table most of the attributes of all _TimeSeries subclasses are merged. The table contains the attribute OBJECTCLASS_ID which references table OBJECTCLASS and allows to distinguish between the four type of times series. It also contains the GMLID and GMLID_CODESPACE attributes, plus the attributes from the TimeValuesProperties class. The next free ID value is provided by the database sequence NRG_TIMESERIES_ID_SEQ. Regular time series must be stored as an array of values in column VALUES_ARRAY. The units of measure are stored in the corresponding attribute VALUES_UNIT. For the irregular time series, the same approach still holds, but the vector of values is stored together with an array of ordered timestamps in the attribute TIMESTAMP_VALUES.

Please note that the adoption of an array type for the columns VALUES_ARRAY and TIME_ARRAY is PostgreSQL-specific and does not follow the 3DCityDB implementation rules which convert multiple occurrences of the same element into a string, where the values are separated by the ‘--/\--‘ string sequence (e.g. with attributes FUNCTION and USAGE). However, in this (only) case, this implementation decision was made in order to take advantage of the specific array functions already available in PostgreSQL.

Please note that, in order to input an array in PostgreSQL, two main methods exist. In the case of the VALUES_ARRAY and TIME_ARRAY columns, 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])

The remaining attributes for file-based time series are merged in the table **NRG_TIMESERIES_FILE**, which is linked to the NRG_TIMESERIES table by means of the foreign key ID. Additionally, the Boolean attribute IS_COMPRESSED is provided, although not included in the original UML diagrams.

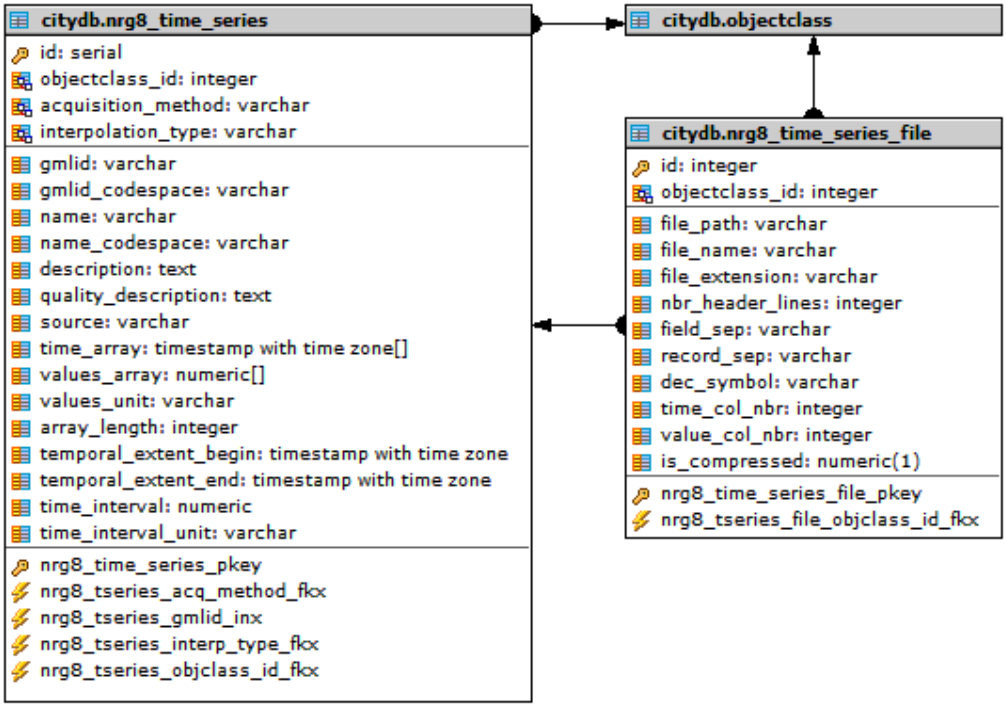


Figure 13: ER model for time series

When it comes to the _Schedule class and the related/derived classes, three tables are created. Please refer to Figure 14 for the representation of the corresponding ER diagram.

In general, the **NRG_SCHEDULE** table merges all attributes of all four _Schedule subclasses. The table contains the attribute OBJECTCLASS_ID which references table OBJECTCLASS and allows to distinguish between the four type of schedules. It also contains the GMLID and GMLID_CODESPACE attributes. The next free ID value is provided by the database sequence NRG_SCHEDULE_ID_SEQ. The VALUE1 attribute (and the corresponding VALUE1_UNIT) are used to store either the averageValue attribute of ConstantValueSchedule class, or the idleValue attribute of DualValueSchedule class, while VALUE2 is used for the usageValue attribute.

For the TimeSeriesSchedule class, the foreign key TIME_SERIES_ID references the NRG_TIME_SERIES table. For the DailyPatternSchedule class, two more tables are created. The **NRG_PERIOD_OF_YEAR** table, linked to the NRG_SCHEDULE table by the TIME_SERIES_ID foreign key, and the **NRG_DAILY_SCHEDULE**, which is connected to the NRG_PERIOD_OF_YEAR and the NRG_TIMESERIES tables by means of the PERIOD_OF_YEAR_ID and TIME_SERIES_ID respectively. For both tables NRG_PERIOD_OF_YEAR and NRG_DAILY_SCHEDULE, the next free ID values are provided by the respective database sequences NRG_PERIOD_OF_YEAR_ID_SEQ and NRG_DAILY_SCHEDULE_ID_SEQ.

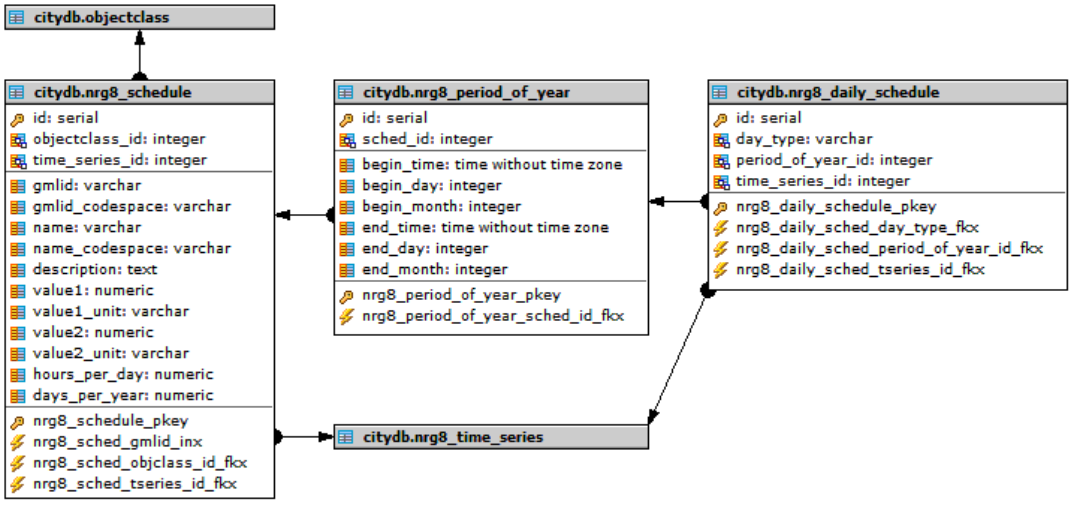


Figure 14: ER model for schedules

3.2.2 Material and Construction module

For the Material and Construction module, five tables are created. Please refer to Figure 15 for the representation of the corresponding ER diagram.

The **NRG_CONSTRUCTION** table merges the attributes of the Construction and ReverseConstruction classes. It contains the GMLID (+ GMLID_CODESPACE), the NAME and DESCRIPTION attributes. The next free ID value is provided by the database sequence NRG_CONSTRUCTION_ID_SEQ. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between Construction and ReverseConstruction. In particular, ReverseConstruction objects can store the ID of the referenced base construction in the BASE_CONSTR_ID foreign key, which references the ID of the same table.

In the case of a Construction class, the u-Value, the service-of-live-related attributes, and the glazingRatio attribute from the OpticalProperties are stored directly in the CONSTRUCTION table. For all remaining optical properties, data is stored in the NRG_OPTICAL_PROPERTY table (see later). The service-of-live-related attributes are stored directly in the table.

In order to associate a construction to a _CityObject, the table **NRG_CITYOBJECT_TO_CONSTR** is used. The foreign keys CITYOBJECT_ID and CONSTRUCTION_ID refer to the respective tables. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

Non-v.0.8 Implementation: with regards to the Energy ADE, this is actually a relaxation of the relation, as an AbstractConstruction is – as of v. 0.8. – referenced only by a ThermalBoundary or ThermalOpening class. The current implementation allows instead to associate a construction to *any* _CityObject.

In the **NRG_OPTICAL_PROPERTY**, all properties of classes Reflectance, Emissivity and Transmittance are merged. The next free ID value is provided by the database sequence NRG_OPTICAL_PROPERTY_ID_SEQ. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between the different optical properties. Besides the SURF_SIDE and FRACTION attributes, the RANGE one stands for the wavelengthRange attribute. The foreign key CONSTR_ID references the table NRG_CONSTRUCTION.

The **NRG_LAYER** table corresponds to the Layer class. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The next free ID value is provided by the database sequence NRG_LAYER_ID_SEQ. The POS_NBR attribute is used to store the position of the layer in the whole construction with regards to the other layers. This attribute is of type integer. The foreign key CONSTR_ID references table NRG_CONSTRUCTION.

The **NRG_LAYER_COMPONENT** table corresponds to the LayerComponent class. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_LAYER_COMPONENT_ID_SEQ. The service-of-live-related attributes are stored directly in the table. The foreign key LAYER_ID references the table NRG_LAYER.

The **NRG_MATERIAL** table corresponds to the _Material class and merges all attributes of its two subclasses. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between the different material classes. The next free ID value is provided by the database sequence NRG_MATERIAL_ID_SEQ. The foreign key LAYER_COMPONENT_ID references table NRG_LAYER_COMPONENT.

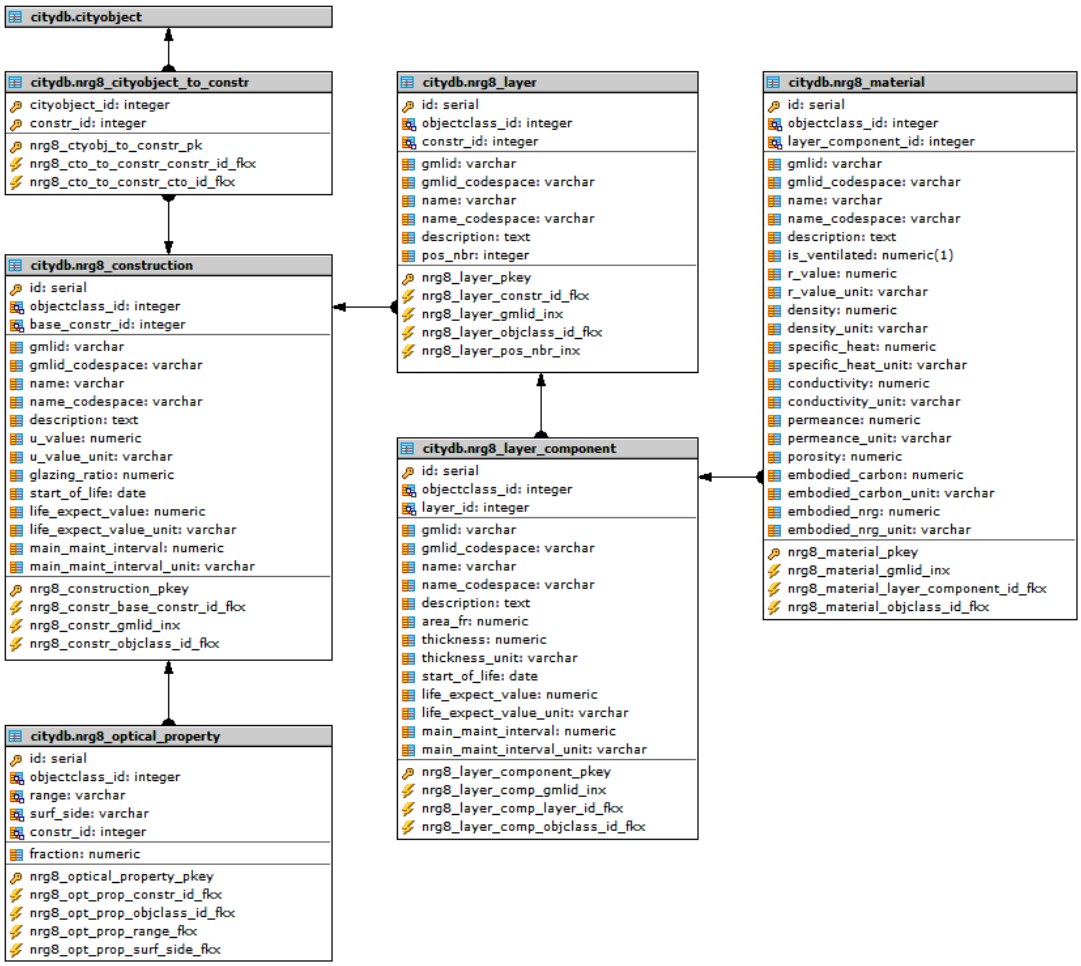


Figure 15: ER model for Material and Construction module

3.2.3 Building Physics module

For the BuildingPhysics module, several tables are created. Please refer to Figure 16 for the representation of the corresponding ER diagram.

First of all, some "multi-purpose" ancillary tables, i.e. the NRG_PERF_CERTIFICATION, the NRG_REFURBISHMENT_MEASURE and the NRG_DIMENSIONAL_ATTRIB are created.

The **NRG_PERF_CERTIFICATION** table corresponds to the EnergyPerformanceCertification class. The next free ID value is provided by the database sequence NRG_PERF_CERTIFICATION_ID_SEQ, while the remaining attributes (NAME, RATING, CERTIFICATION_ID) correspond to those in the class. The foreign keys BUILDING_ID and BUILDING_UNIT_ID refer to the tables NRG_BUILDING and NRG_BUILDING_UNIT, respectively (see later for details).

The **NRG_REFURBISHMENT_MEASURE** table corresponds to the RefurbishmentMeasure class. The next free ID value is provided by the database sequence

NRG_REFURBISHMENT_MEASURE_ID_SEQ. The INSTANT_DATE attribute must be used to store the date of the event, while in case of a time period, the two attributes BEGIN_DATE and END_DATE have both to be used. The foreign keys BUILDING_ID and THERM_BOUNDARY_ID refer to the tables NRG_BUILDING and NRG_THERMAL_BOUNDARY, respectively (see later for details).

The **NRG_DIMENSIONAL_ATTRIB** serves to store the attributes of classes HeightAboveGround, FloorArea and Volume. The next free ID value is provided by the database sequence NRG_DIMENSIONAL_ATTRIB_ID_SEQ. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between the different dimensional attributes. The TYPE attribute is used to store the values of the codelists or enumerations of the respective attribute (e.g. NetVolume, GrossVolume, EnergyReferenceVolume for Volume). The VALUE attribute stores the actual value. The CITYOBJECT_ID foreign key references the CITYOBJECT table.

The **NRG_WEATHER_DATA** table corresponds to the WeatherData class. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The next free ID value is provided by the database sequence NRG_WEATHER_DATA_ID_SEQ. The TYPE attribute corresponds to the weatherDataType attribute, while the INSTALL_POINT corresponds to position attribute in the WeatherData class and contains a POINTZ-geometry. The foreign keys CITYOBJECT_ID and TIME_SERIES_ID refer to the tables CITYOBJECT and NRG_TIME_SERIES, respectively.

The **NRG_WEATHER_STATION** table corresponds to the WeatherStation class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The INSTALL_POINT attribute corresponds to position attribute in the WeatherStation class and contains a POINTZ-geometry. The foreign key CITYOBJECT_ID refers to the table CITYOBJECT.

Non-v.0.8 Implementation: The NRG_WEATHER_STATION table implements the WeatherStation class, which is going to be included in the Energy ADE v. 0.9. A WeatherStation is a class derived by a _CityObject class which is associated to 0..* WeatherData classes. Besides, a WeatherStation class has a position attribute of type GM_Point with cardinality [0..1]. For more details, please refer GitHub ticket #135, https://github.com/cstb/citygml-energy/issues/135.

The **NRG_BUILDING** table extends the 3DCityDB “vanilla” BUILDING table and corresponds to the ADE-related _AbstractBuilding class. It is linked to the BUILDING table by means of the foreign key ID, which acts also as primary key. It contains the TYPE (+TYPE_CODESPACE), CONSTR_WEIGHT, IS_LANDMARKED and the POINTZ-geometry REF_POINT attributes which correspond to buildingType, constructionWeight, isLandmarked, and referencePoint ones. The remaining attributes with cardinality 0..* are stored in the previously described ancillary tables (NRG_PERF_CERTIFICATION, NRG_REFURBISHMENT_MEASURE and NRG_DIMENSIONAL_ATTRIB).

The **NRG_THERMAL_ZONE** table corresponds to class ThermalZone. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. As mentioned before, the attributes with cardinality 0..* are stored by means of ancillary tables (here: NRG_DIMENSIONAL_ATTRIB). The foreign key SOLID_ID refer to the SURFACE_GEOMETRY table where the volumeGeometry data are to be stored according to the rules defined and described already in the 3DCityDB documentation. For convenience and for testing purposes, a MULTI_SURF_GEOM attribute is also currently available which allows for storage of MULTIPOLYGONZ geometries directly inline. Please note that the MULTI_SURF_GEOM column is going to be removed in future releases. In addition, geometries stored here cannot be associated to any CityGML appearance feature.

By means of the table **NRG_THERMAL_ZONE_TO_ROOM** the relation between the ROOM and the NRG_THERMAL_ZONE tables is established, in that the foreign keys THERM_ZONE_ID and ROOM_ID refer to the respective tables. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

A similar approach is followed also for the tables NRG_THERMAL_BOUNDARY and NRG_THERMAL_OPENING, respectively, with the attributes stored directly inline or in ancillary tables.

In particular, the **NRG_THERMAL_BOUNDARY** table is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The TYPE attribute corresponds to the thermalBoundaryType one, the reference to the staticConstruction is stored in the linked NRG_CITYOBJECT_TO_CONSTR table, and for the surfaceGeometry the MULTI_SURF_ID foreign key refer to table SURFACE_GEOMETRY. Given the relevance of the order for shared walls delimiting two thermal zones, two columns (THERMAL_ZONE1_ID and THERMAL_ZONE2_ID) are used to store the ID of the parent object(s) from the NRG_THERMAL_ZONE table. For convenience and for testing purposes, a MULTI_SURF_GEOM attribute is also currently available which allows for storage of MULTIPOLYGONZ geometries directly inline. The same applies here as stated for the MULTI_SURF_GEOM column in table NRG_THERMAL_ZONE.

The **NRG_THERM_BDRY_TO_THEM_SURF** table stores the association between ThermalBoundary and _BoundarySurface classes. The implementation logic follows the one seen for the NRG_THERMAL_ZONE_TO_ROOM. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

The **NRG_THERMAL_OPENING** table corresponds to the ThermalOpening class, however it includes also the attributes of the ShadingType class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The foreign keys THERM_BOUNDARY_ID and MULTI_SURF_ID refer to tables NRG_THERMAL_BOUNDARY and GEOMETRY_SURFACE, respectively. For convenience and for testing purposes, a MULTI_SURF_GEOM attribute is also currently available which allows for storage of MULTIPOLYGONZ geometries directly inline. The same applies here as stated for the MULTI_SURF_GEOM column in table NRG_THERMAL_ZONE.

The **NRG_THERMAL_OPEN_TO_OPEN** table stores the relation between the ThermalOpening and the _Opening class. The implementation logic follows the one seen for the NRG_THERMAL_ZONE_TO_ROOM. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

Please note that this implementation already allows for a 0..* cardinality (as in GitHub ticket #133, https://github.com/cstb/citygml-energy/issues/133), instead of the 0..1 as in the Energy ADE 0.8.

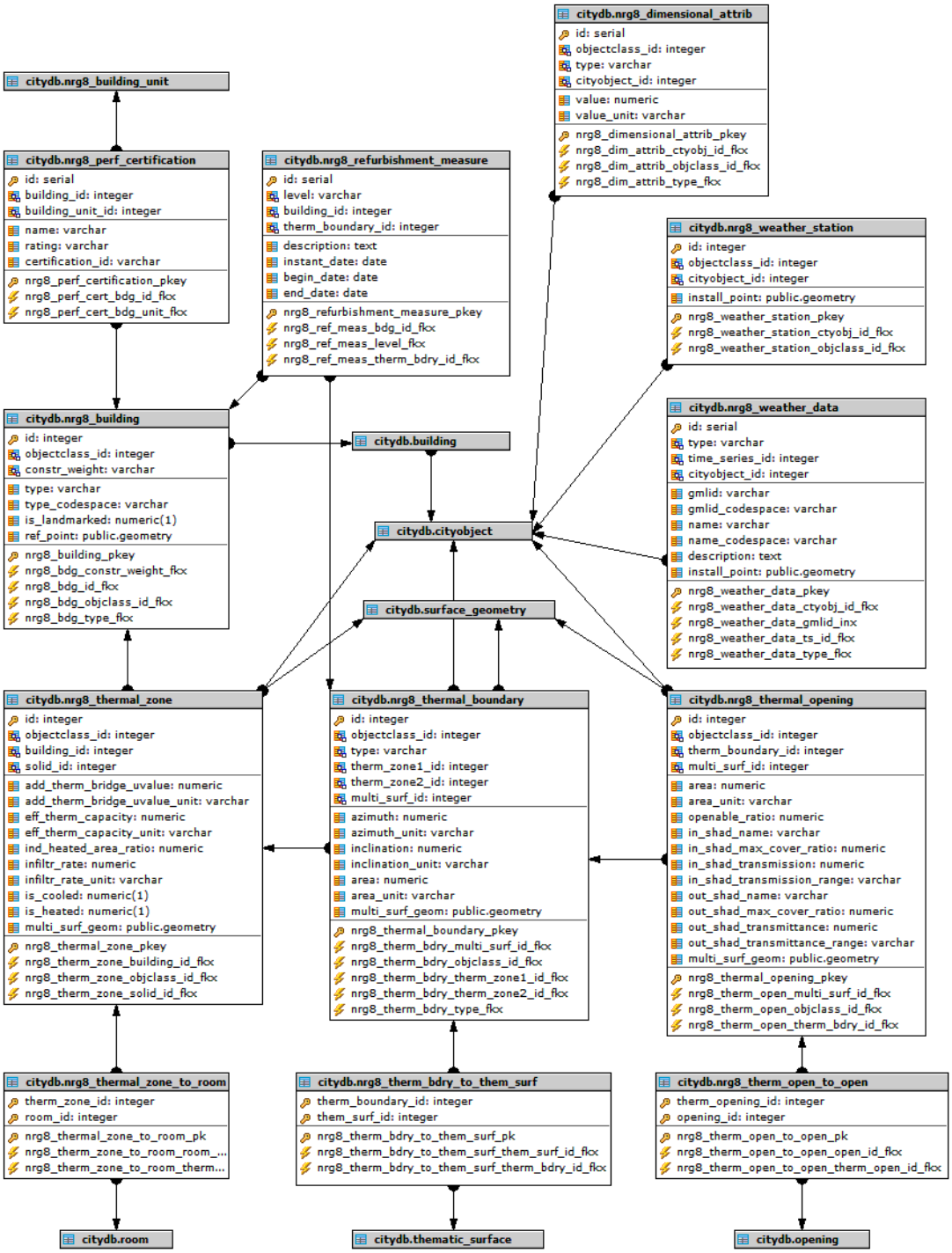


Figure 16: ER model for Building Physics module

3.2.4 Occupancy module

For the Occupancy module, several design strategies described in the previous paragraph still apply, so they will not be explained in detail again. Please refer to Figure 17 for the representation of the corresponding ER diagram.

The **NRG_USAGE_ZONE** table corresponds to the UsageZone class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The TYPE (+ TYPE_CODESPACE) attributes correspond to the usageZoneType one, and all heat-exchange-related attributes are also stored directly in the table. The ancillary table NRG_DIMENSIONAL_ATTRIB is used to store the floorArea attribute. The foreign keys COOL_SCHED_ID, HEAT_SCHED_ID and VENT_SCHED_ID refer to table NRG_SCHEDULE. The foreign key THERM_ZONE_ID refers to table NRG_THERMAL_ZONE. The foreign key SOLID_ID refers to the SURFACE_GEOMETRY table. For convenience and for testing purposes, a MULTI_SURF_GEOM attribute is also currently available which allows for storage of MULTIPOLYGONZ geometries directly inline. The same applies here as stated for the MULTI_SURF_GEOM column in table NRG_THERMAL_ZONE.

The **NRG_BUILDING_UNIT** table corresponds to the BuildingUnit class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The floorArea and the energyPerformanceCertification attributes are stored in the ancillary tables NRG_DIMENSIONAL_ATTRIB and NRG_PERF_CERTIFICATION. The foreign key USAGE_ZONE_ID refers to table NRG_USAGE_ZONE.

Table **NRG_BDG_UNIT_TO_ADDRESS** stores the relations between table NRG_BUILDING_UNIT and ADDRESS. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

Table **NRG_FACILITIES** corresponds to class _Facilities and merges its attributes and those of the subclasses, as well as all heat-exchange-related ones. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between the different facilities. All heat-exchange-related attributes are also stored directly in the table. The foreign key OPER_SCHED_ID refers to table NRG_SCHEDULE, while the foreign keys BUILDING_UNIT_ID and USAGE_ZONE_ID refer to tables NRG_BUILDING_UNIT and NRG_USAGE_ZONE_ID, respectively.

Non-v.0.8 Implementation: Class _Facilities is considered to be a subclass of _CityObject, therefore the implementation follows the rules for _CityObject's derived classes. For more details, please refer to GitHub ticket #132, https://github.com/cstb/citygml-energy/issues/132.

Table **NRG_OCCUPANTS** corresponds to class Occupants. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_ NRG_OCCUPANTS_ID_SEQ. The TYPE attribute corresponds to the occupancyType one. All heat-exchange-related attributes are also stored directly in the table. The foreign key OCCUP_SCHED_ID refers to table NRG_SCHEDULE, while the foreign keys BUILDING_UNIT_ID and USAGE_ZONE_ID refer to tables NRG_BUILDING_UNIT and NRG_USAGE_ZONE_ID, respectively.

Table **NRG_HOUSEHOLD** corresponds to class Household. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described.

The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_HOUSEHOLD_ID_SEQ. Attribute TYPE corresponds to householdType attribute. The foreign key OCCUPANTS_ID refers to table NRG_OCCUPANTS.

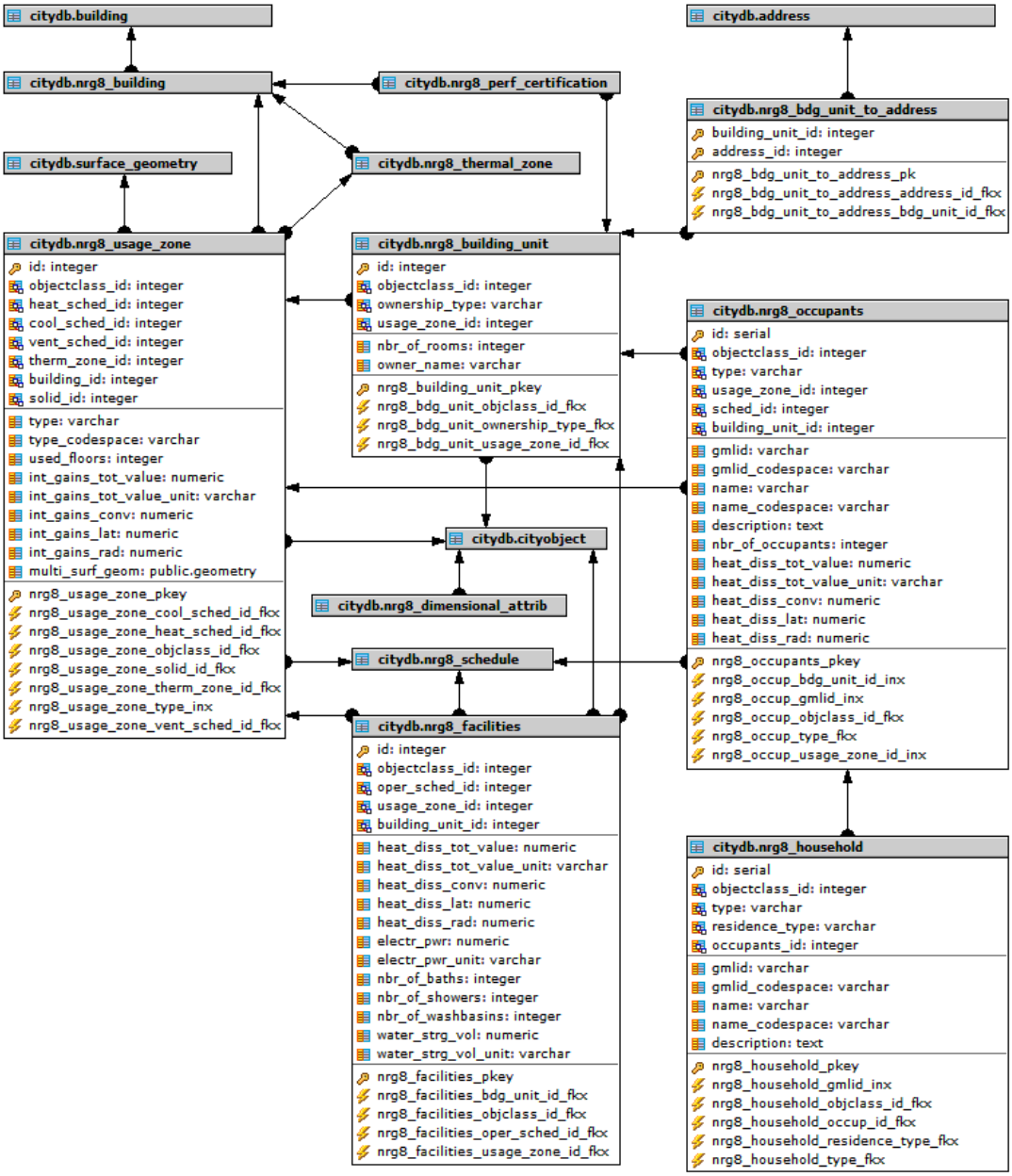


Figure 17: ER model for Occupancy module

3.2.5 Energy Systems module

For the Energy Systems module, several design strategies described in the previous paragraphs still apply, so they will not be explained in detail again. Please refer to Figure 18 and Figure 19 for the representation of the corresponding ER diagrams.

The **NRG_ENERGY_DEMAND** table corresponds to the EnergyDemand class. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_ENERGY_DEMAND_ID_SEQ. The foreign keys CITYOBJECT_ID and TIME_SERIES_ID refer to tables CITYOBJECT and NRG_TIME_SERIES, respectively.

Non-v.0.8 Implementation: Please note that classes EnergyDistributionSystem, Emitter, _EnergyStorageSystem, EnergyConversionSystem are all considered to be derived from _CityObject, therefore some attributes are stored in the CITYOBJECT table. This change reflects what will happen in v. 0.9 (as in GitHub ticket #131, https://github.com/cstb/citygml-energy/issues/131). As a consequence, the attribute productAndInstallationDocument of class EnergyConversionSystem is stored directly in table EXTERNAL_REFERENCE.

Non-v.0.8 Implementation: Please note that classes EnergyDistributionSystem, Emitter, _EnergyStorageSystem are extended by means of adding to each of them an association conceptually similar to the energyConversionSystem from _CityObject to EnergyConversionSystem (see UML diagram of the Energy Systems module). These associations are all implemented in the same way by means of a foreign key from the respective table to table CITYOBJECT. This change reflects what will happen in v. 0.9 (as in GitHub ticket #137, https://github.com/cstb/citygml-energy/issues/137).

The **NRG_STORAGE_SYSTEM** table corresponds to the _StorageSystem class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. All service-of-life related attributes are stored directly in the table. The foreign keys NRG_DEMAND_ID and CITYOBJECT_ID refers to tables NRG_ENERGY_DEMAND and CITYOBJECT, respectively.

Tables **NRG_THERMAL_STORAGE_SYSTEM** and **NRG_POWER_STORAGE_SYSTEM** correspond to classes ThermalStorageSystem and PowerStorageSystem. Both are linked to the NRG_STORAGE_SYSTEM table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The two tables contain the attributes of the corresponding class, respectively.

The **NRG_DISTRIB_SYSTEM** table corresponds to the DistributionSystem class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. All service-of-life related attributes are stored directly in the table. The foreign keys NRG_DEMAND_ID and CITYOBJECT_ID refers to tables NRG_ENERGY_DEMAND and CITYOBJECT, respectively.

Tables **NRG_THERMAL_DISTRIB_SYSTEM** and **NRG_POWER_DISTRIB_SYSTEM** correspond to classes ThermalDistributionSystem and PowerDistributionSystem. Both are linked to the NRG_DISTRIB_SYSTEM table by means of the foreign key ID, which acts also

as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The two tables contain the attributes of the corresponding class, respectively.

The **NRG_EMITTER** table corresponds to the Emitter class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. All thermal-exchange-related attributes are stored directly in the table. The foreign keys DISTRIB_SYSTEM_ID and CITYOBJECT_ID refer to tables NRG_DISTRIB_SYSTEM and CITYOBJECT, respectively.

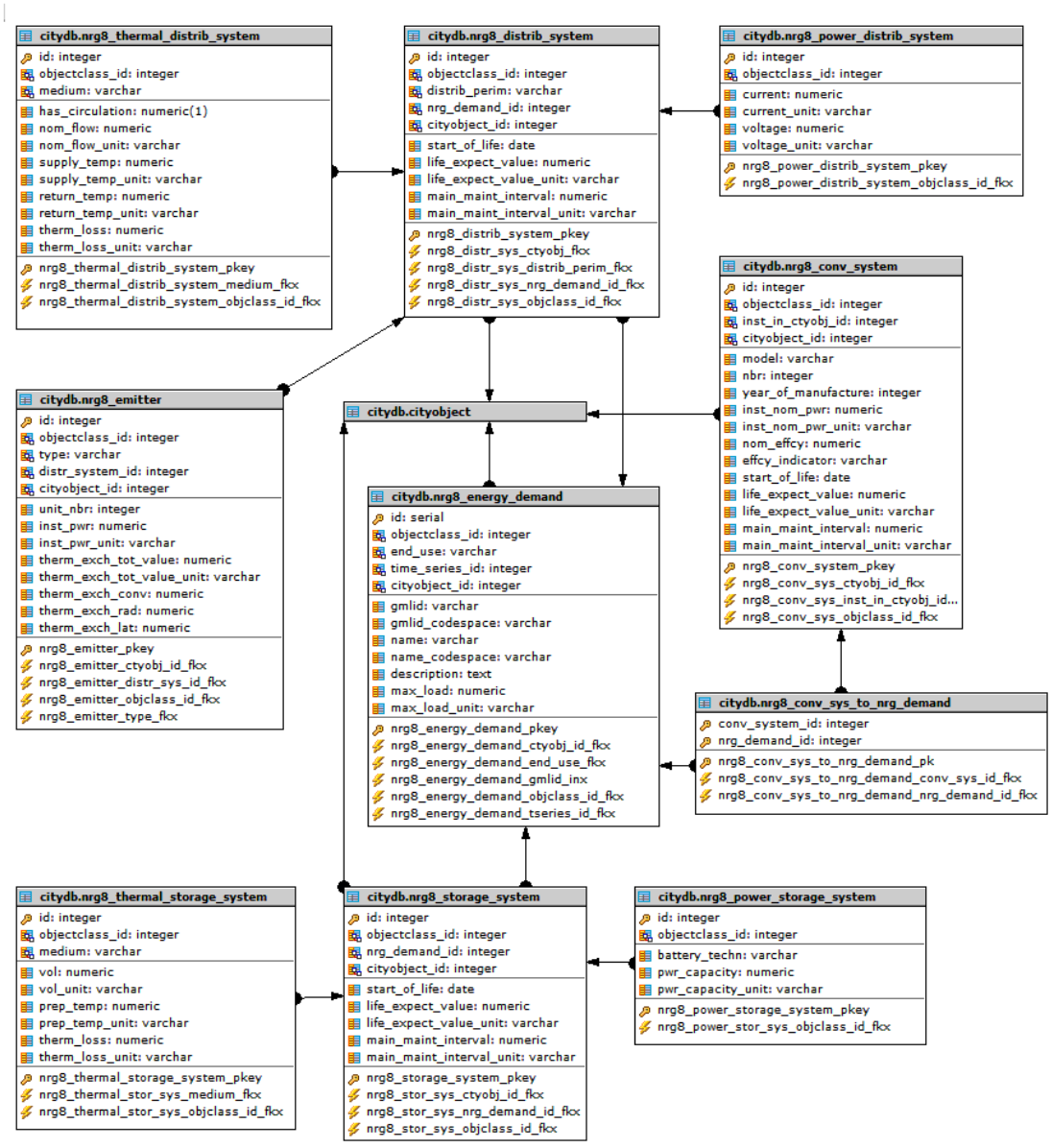


Figure 18: ER model for Energy Systems module (part A)

The **NRG_FINAL_ENERGY** table corresponds to the FinalEnergy class. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously

described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_FINAL_ENERGY_ID_SEQ. All energy-carrier-related attributes are stored directly in the table. The foreign key TIME_SERIES_ID refers to tables NRG_TIME_SERIES.

The **NRG_SYSTEM_OPERATION** table corresponds to class SystemOperation. It contains the ID, GMLID, GMLID_CODESPACE, NAME, DESCRIPTION attributes as already previously described. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The next free ID value is provided by the database sequence NRG_SYSTEM_OPERATION_ID_SEQ. The foreign keys SCHED_ID and NRG_CONV_SYSTEM_ID refer to tables NRG_SCHEDULE and NRG_CONV_SYSTEM, respectively.

The **NRG_CONV_SYSTEM** table corresponds to the EnergyConversionSystem and ElectricalResistance class. It is linked to the CITYOBJECT table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish between the different conversion systems. All service-of-life-related attributes are stored directly in the table. The foreign keys CIYOBJECT_ID and INST_IN_CTYOBJ_ID refer both to table CITYOBJECT. The former corresponds to the energyConversionSystem attribute, the latter the installedIn one.

The **NRG_SOLAR_SYSTEM** table merges class _SolarEnergySystem and all subclasses (SolarThermalSystem, PhotovoltaicSystem, PhotovoltaicThermalSystem). It is linked to the NRG_CONV_SYSTEM table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS and allows to distinguish the different solar systems devices. The foreign keys THEM_SURF_ID and BUILDING_INST_ID refer to tables THEMATIC_SURFACE and BUILDING_INSTALLATION, respectively. The foreign key MULTI_SURF_ID refers to the SURFACE_GEOMETRY table. For convenience and for testing purposes, a MULTI_SURF_GEOM attribute is currently also available which allows for storage of MULTIPOLYGONZ geometries directly inline. The same applies here as stated for the MULTI_SURF_GEOM column in table NRG_THERMAL_ZONE.

Non-v.0.8 Implementation: The type of attribute eta0 is considered as scale, instead of decimal. This change reflects what is expected to happen in v. 0.9 (as in GitHub ticket #134, https://github.com/cstb/citygml-energy/issues/134).

The tables **NRG_BOILER**, **NRG_MECH_VENTILATION**, **NRG_HEAT_PUMP**, **NRG_COMBINED_HEAT_POWER**, **NRG_HEAT_EXCHANGER**, **NRG_CHILLER**, **NRG_AIR_COMPRESSOR** correspond to the Boiler, MechanicalVentilation, HeatPump, CombinedHeatPower, HeatExchanger, Chiller and AirCompressor classes, respectively. They are all conceptually and structurally similar. They are linked to the NRG_CONV_SYSTEM table by means of the foreign key ID, which acts also as primary key. The attribute OBJECTCLASS_ID references the table OBJECTCLASS. The tables contain the attributes of the corresponding classes, respectively.

Finally, the **NRG_CONV_SYSTEM_TO_NRG_DEMAND** and the **NRG_CONV_SYSTEM_TO_FINAL_NRG** tables realise the n:m associations between EnergyConversionSystem and EnergyDemand and between EnergyConversionSystem and FinalEnergy classes, respectively. Table NRG_CONV_SYSTEM_TO_FINAL_NRG has an additional attribute ROLE which must contains either the value ‘production’ or ‘consumption’, therefore allowing to represent the different semantics of the associations. Upon delete and update of the referenced tables, the corresponding record is deleted or updated automatically.

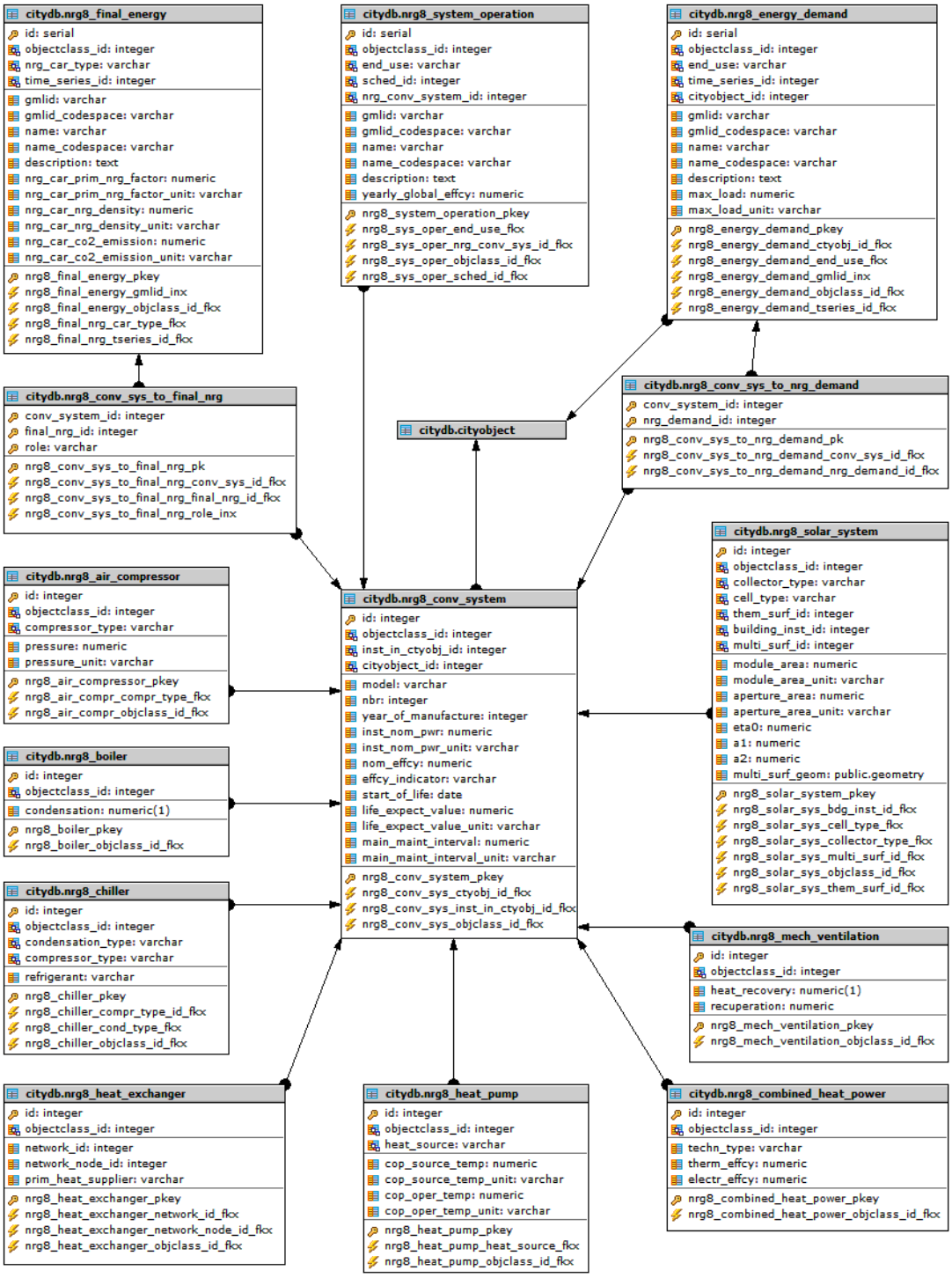


Figure 19 ER model for Energy Systems module (part B)

3.2.6 Lookup tables

A number of lookup tables is also provided. They implement enumerations or codelists defined in the Energy ADE. All tables are stored in the *citydb* schema and are named by

adding the "nrg_lu_" prefix to each table. Table 2 contains the list of the lookup tables shipped with the current implementation of the Energy ADE and the class they correspond to.

LOOKUP TABLE	ENERGY ADE CLASS	REFERENCING TABLE(S)
nrg_lu_acquisition_method	AcquisitionMethodValue	nrg_time_series
nrg_lu_cell	CellTypeValue	nrg_solar_system
nrg_lu_collector	CollectorTypeValue	nrg_solar_system
nrg_lu_compressor	CompressorTypeValue	nrg_air_compressor, nrg_chiller
nrg_lu_condensation	CompressorTypeValue	nrg_chiller
nrg_lu_construction_weight	ConstructionWeightValue	nrg_building
nrg_lu_day	DayTypeValue	nrg_daily_schedule
nrg_lu_dimensional_attrib	OfficialAreaReferenceValue, ElevationReferenceValue, VolumeTypeValue	nrg_building, nrg_thermal_zone, nrg_usage_zone, nrg_building_unit
nrg_lu_distribution	DistributionTypeValue	nrg_distrib_system
nrg_lu_emitter	EmitterTypeValue	nrg_emitter
nrg_lu_end_use	EndUseTypeValue	nrg_energy_demand, nrg_system_operation
nrg_lu_energy_source	CompressorTypeValue	nrg_final_energy
nrg_lu_heat_source	HeatSourceTypeValue	nrg_heat_pump
nrg_lu_household	HouseholdTypeValue	nrg_household
nrg_lu_interpolation	InterpolationTypeValue	nrg_time_series
nrg_lu_medium	MediumTypeValue	nrg_thermal_distrib_system, nrg_thermal_storage_system
nrg_lu_occupants	OccupantTypeValue	nrg_occupants
nrg_lu_ownership	OwnershipTypeValue	nrg_building_unit
nrg_lu_refurbishment_class	RefurbishmentClassValue	nrg_refurbishment_measure
nrg_lu_residence	ResidenceTypeValue	nrg_household
nrg_lu_surface_side	SurfaceSide	nrg_optical_property
nrg_lu_thermal_boundary	ThermalBoundaryTypeValue	nrg_thermal_boundary
nrg_lu_wavelength_range	WavelengthRangeType	nrg_optical_property
nrg_lu_weather_data	WeatherDataTypeValue	nrg_weather_data

Table 2: Lookup tables shipped with this implementation of the Energy ADE. All lookup tables are stored in schema *citydb* and prefixed with *"nrg_lu_"*

3.2.7 Sequences

Upon installation, a number of database sequences is also generated. They can be identified by means of the db_prefix (e.g. "nrg_"). Analogously to other 3DCityDB tables, it is highly recommended to generate ID values for the respective Energy ADE tables by using the predefined sequences only.

Please note that, unlike in the 3DCityDB SQL scripts, all new sequences are generated directly within the CREATE TABLE statement as follows:

```
CREATE TABLE nrg_time_series (
id serial PRIMARY KEY,
name varchar,
…);
```

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

```
CREATE SEQUENCE nrg_time_series_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
ALTER TABLE nrg_time_series_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

3.2.8 Stored procedures

The 3DCityDB extension for the Energy ADE 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 *citydb_pkg*. Coherently with the naming rules, all stored procedures are prefixed with the db_prefix (e.g. "nrg"). Please note that, as already mentioned, the input parameters are omitted and indicated by a simple "…" in the following text for better readability.

At the moment, two main families of stored procedures are shipped. They can be generally grouped into the "delete" and "get_envelope" stored procedures. Some additional miscellaneous stored procedures are also provided, as they are required mostly by other stored procedures belonging to the Metadata module.

A "delete" stored procedure allows to delete an Energy ADE object and all its dependences at once. All stored procedures of this kind are named `nrg_delete_*(…)`, where * is generally the corresponding name of the object to be deleted from the database. So, `nrg_delete_thermal_zone(thermal_zone_id)` will delete all data related to the thermal zone having as ID the `thermal_zone_id` (e.g. 22456). Conceptually, these stored procedures reflect the analogous ones shipped with the "vanilla" 3DCityDB.

The second family of stored procedures is named `nrg_get_envelope_*(…)` where * is generally the corresponding name of the object. These stored procedures allows to compute the 3D envelope of those ADE objects which have geometries and conceptually reflect the analogous ones shipped with the "vanilla" 3DCityDB.

Please note that each set of stored procedures is conceived to work exclusively with objects of the corresponding ADE. Therefore, a nrg_delete_* stored procedures will work only with objects of the specific ADE (in this case, the Energy ADE using the db_prefix "nrg"). However, the Metadata module also delivers two improved and ADE-enabled high hierarchy stored procedures, namely `delete_cityobject(…)` and `get_envelope_cityobject(…)` which now have the advantage to work with *any* object (both the "vanilla" ones and those of *any* ADE). These two new "delete" and "get_envelope" stored procedures require exactly same input parameters of the "vanilla" ones.

Please note: the "vanilla" 3DCityDB also provides stored procedures to delete multiple objects at a time by means of the lineage column in the CITYOBJECT table. Such stored procedures are named with plural names, e.g. `delete_buildings(…)` or `delete_cityobjectgroups(…)` (please note the **s** at the end of the names).

The current implementation of the Energy ADE still lacks such lineage-based stored procedures, they will be added in future releases.

In Table 3 all stored procedures shipped with this implementation of the Energy ADE are listed.

DELETE STORED PROCEDURES
nrg_delete_building(…)
nrg_delete_building_unit(…)
nrg_delete_cityobject(…)
nrg_delete_construction(…)
nrg_delete_conv_system(…)
nrg_delete_daily_schedule(…)
nrg_delete_distrib_system(…)
nrg_delete_emitter(…)
nrg_delete_energy_demand(…)
nrg_delete_facilities(…)
nrg_delete_final_energy(…)
nrg_delete_household(…)
nrg_delete_layer(…)
nrg_delete_layer_component(…)
nrg_delete_material(…)
nrg_delete_occupants(…)
nrg_delete_period_of_year(…)
nrg_delete_schedule(…)
nrg_delete_storage_system(…)
nrg_delete_system_operation(…)
nrg_delete_thermal_boundary(…)
nrg_delete_thermal_opening(…)
nrg_delete_thermal_zone(…)
nrg_delete_time_series(…)
nrg_delete_usage_zone(…)
nrg_delete_weather_data(…)
nrg_delete_weather_station(…)
GET ENVELOPE STORED PROCEDURES
nrg_get_envelope_cityobject(…)
nrg_get_envelope_conv_system(…)
nrg_get_envelope_thermal_boundary(…)
nrg_get_envelope_thermal_opening(…)
nrg_get_envelope_thermal_zone(…)
nrg_get_envelope_usage_zone(…)
MISCELLANEAOUS STORED PROCEDURES
nrg_intern_delete_cityobject(…)
nrg_has_no_reverse_construction(…)
nrg_set_ade_columns_srid(…)
nrg_cleanup_schema(…)

Table 3: Stored procedures shipped with this implementation of the Energy ADE. All stored procedures are stored in schema *citydb_pkg*.

3.2.9 Views

Please note: the principles behind the database objects described in this section follow conceptually what is described in **Appendix A** about the **3DCityDB Utilities Package**. Please refer to that section for a more detailed documentation of characteristics and implementation properties.

As described in section 3.1.2 (Mapping rules for ADE classes), 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[3] operations (i.e. select, insert, update and delete), especially when dealing with the tables added by an ADE (and as long the Importer/Exporter does not support ADEs natively).

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.

For example, the view named **NRG_BUILDING** consists of the joined tables CITYOBJECT, BUILDING and NRG_BUILDING and presents data as a single table. 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 view names are prefixed with the *db_prefix*. All views are installed in a specific, newly added, database schema named *citydb_view*.

Please note: in order to access *any* database object in the *citydb_view* schema, **qualified names**[4] consisting of the schema name and the object name separated by a dot ***must*** **be used**. This is a design decision, in order to avoid homonymy issues since some objects (e.g. views) have the same names of other objects in the *citydb* and *citydb_pkg* schemas. 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 4 contains all views shipped with this version of the 3DCityDB extension for the Energy ADE. An example of the **NRG_BUILDING** and **NRG_THERMAL_ZONE** views is presented in Figure 21: and Figure 20, respectively.

VIEW	UPDATABLE?
nrg_building	✓
nrg_building_part	✓
nrg_building_unit	✓
nrg_construction	
nrg_construction_base	✓
nrg_construction_reverse	✓
nrg_conv_system_generic	✓
nrg_conv_system_air_compressor	✓

[3] CRUD represents an acronym for the database operations **C**reate, **R**ead, **U**pdate, and **D**elete.
[4] For more details, see https://www.postgresql.org/docs/9.1/static/ddl-schemas.html

VIEW	UPDATABLE?
nrg_conv_system_boiler	✓
nrg_conv_system_chiller	✓
nrg_conv_system_combined_heat_power	✓
nrg_conv_system_electrical_resistance	✓
nrg_conv_system_heat_exchanger	✓
nrg_conv_system_heat_pump	✓
nrg_conv_system_mech_ventilation	✓
nrg_conv_system_solar_pv	✓
nrg_conv_system_solar_pv_thermal	✓
nrg_conv_system_solar_thermal	✓
nrg_daily_schedule	✓
nrg_daily_schedule_ts	✓
nrg_dimensional_attrib	
nrg_dimensional_attrib_floor_area	✓
nrg_dimensional_attrib_height	✓
nrg_dimensional_attrib_volume	✓
nrg_distrib_system_generic	✓
nrg_distrib_system_power	✓
nrg_distrib_system_thermal	✓
nrg_emitter	✓
nrg_energy_demand	✓
nrg_energy_demand_ts	✓
nrg_facilities	
nrg_facilities_dhw	✓
nrg_facilities_electrical_appliances	✓
nrg_facilities_lighting	✓
nrg_final_energy	✓
nrg_final_energy_ts	✓
nrg_household	✓
nrg_layer	✓
nrg_layer_component	✓
nrg_material	
nrg_material_gas	✓
nrg_material_solid	✓
nrg_occupants	✓
nrg_optical_property	
nrg_optical_property_emissivity	✓
nrg_optical_property_reflectance	✓
nrg_optical_property_transmittance	✓
nrg_perf_certification	✓
nrg_period_of_year	✓
nrg_refurbishment_measure	✓
nrg_schedule	
nrg_schedule_constant_value	✓
nrg_schedule_daily_pattern	✓
nrg_schedule_dual_value	✓
nrg_schedule_time_series	✓
nrg_schedule_time_series_ts	✓
nrg_storage_system_power	✓

VIEW	UPDATABLE?
nrg_storage_system_thermal	✓
nrg_system_operation	✓
nrg_thermal_boundary	✓
nrg_thermal_opening	✓
nrg_thermal_zone	✓
nrg_time_series	✓
nrg_time_series_irregular	✓
nrg_time_series_irregular_file	✓
nrg_time_series_regular	✓
nrg_time_series_regular_file	✓
nrg_usage_zone	✓
nrg_weather_data	✓
nrg_weather_data_ts	✓
nrg_weather_station	✓

Table 4: Views shipped with this implementation of the Energy ADE. All views are stored in schema *citydb_view*.

In general, all views follow the same rules of those shipped with the 3DCityDB Utilities Package. In addition to the views, a number of stored procedures, triggers and trigger functions are also installed, both in schema *citydb_pkg* and in schema *citydb_view*. In addition, following notes apply to certain views.

View names with the "*_generic*" suffix (i.e. views **NRG_DISTRIB_SYSTEM_GENERIC**, **NRG_CONV_SYSTEM_GENERIC**) present data of instances of the corresponding class itself, and not of any derived class. In other words, both classes EnergyDistributionSystem and EnergyConversionSystem are not abstract and can thus be instantiated, unlike, for example, _EnergyStorageSystem, which is an abstract class (please refer to UML diagram of the EnergySystem module).

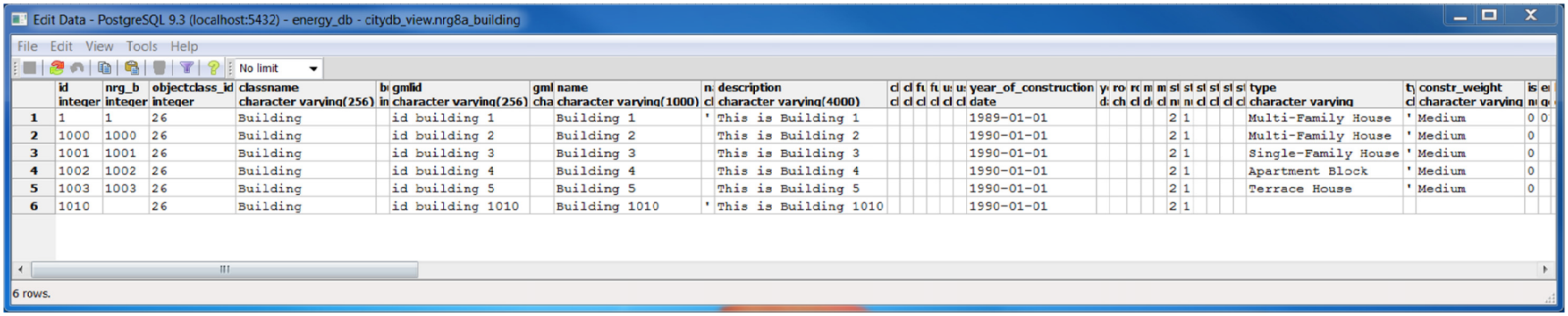
Time series data is presented either in four views specific for each _TimeSeries-derived class according to the general naming rules for the 3DCityDB Utilitites Package, and, additionally, in the view **NRG_TIME_SERIES**, which corresponds to a simple join of the two original tables in schema *citydb*. This updatable view works however in part differently as the others:

- For delete and update operations, standard behaviour as the other views;
- For insert operations, the user *must* declare explicitly the CLASSNAME parameter (i.e. RegularTimeSeries, IrregularTimeSeries, RegularTimeSeriesFile, IrregularTimeSeriesFile) and provide the other proper attributes. Upon insert, the trigger function will take care of retrieving the proper OBJECTCLASS_ID value and to store data accordingly. Regarding the ID parameter, the standard behaviour applies, i.e. the ID can be set by the user or left blank and set automatically then by the insert stored procedures using the first available value from the time_series_id_seq sequence.

Views with suffix "*_ts*" (e.g. **NRG_ENERGY_DEMAND_TS**, **NRG_WEATHER_DATA_TS**, etc.) present data from the equivalent view without suffix in addition to the depending time series data (if available). In other words, such views allow to

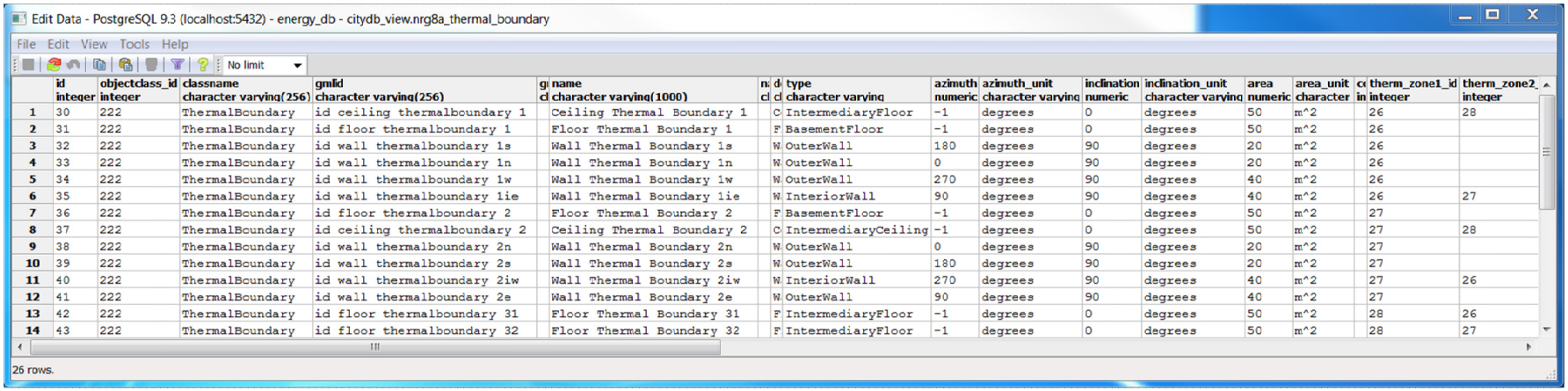
explore the dependent time series data of the respective view at the same time. All columns containing time series data are prefixed with "*ts_*". As these views are also updatable, there are some advantages. Taking the NRG_ENERGY_DEMAND_TS view as example:

1) The user can input the energy demand data *and* the depending time series data at once. The trigger stored procedures will take care of inserting first the time series data and passing the corresponding time series ID parameter (TS_ID) to be stored as reference key with the remaining energy demand data.
2) Like in the case of the NRG_TIME_SERIES view described before, the user *must* declare explicitly the TS_CLASSNAME parameter of the time series.
3) Regarding both the ID and the TS_ID parameters, they can be set by the user or automatically, as usual.
4) For update and delete operations, data in all original tables is updated or deleted automatically. Please note that, by design decision, the OBJECTCLASS_ID and TS_OBJECTCLASS_ID, as well as the ID and TS_ID cannot be changed upon update operations.



	id integer	nrg_b integer	objectclass_id integer	classname character varying(256)	gmlid character varying(256)	name character varying(1000)	description character varying(4000)	year_of_construction date	type character varying	constr_weight character varying
1	1	1	26	Building	id building 1	Building 1	This is Building 1	1989-01-01	Multi-Family House	Medium
2	1000	1000	26	Building	id building 2	Building 2	This is Building 2	1990-01-01	Multi-Family House	Medium
3	1001	1001	26	Building	id building 3	Building 3	This is Building 3	1990-01-01	Single-Family House	Medium
4	1002	1002	26	Building	id building 4	Building 4	This is Building 4	1990-01-01	Apartment Block	Medium
5	1003	1003	26	Building	id building 5	Building 5	This is Building 5	1990-01-01	Terrace House	Medium
6	1010		26	Building	id building 1010	Building 1010	This is Building 1010	1990-01-01		

Figure 21: Example of the NRG_BUILDING view, merging data from tables CITYOBJECT, BUILDING and NRG_BUILDING



	id integer	objectclass_id integer	classname character varying(256)	gmlid character varying(256)	name character varying(1000)	type character varying	azimuth numeric	azimuth_unit character varying	inclination numeric	inclination_unit character varying	area numeric	area_unit character	therm_zone1_id integer	therm_zone2_ integer
1	30	222	ThermalBoundary	id ceiling thermalboundary 1	Ceiling Thermal Boundary 1	IntermediaryFloor	-1	degrees	0	degrees	50	m^2	26	28
2	31	222	ThermalBoundary	id floor thermalboundary 1	Floor Thermal Boundary 1	BasementFloor	-1	degrees	0	degrees	50	m^2	26	
3	32	222	ThermalBoundary	id wall thermalboundary 1s	Wall Thermal Boundary 1s	OuterWall	180	degrees	90	degrees	20	m^2	26	
4	33	222	ThermalBoundary	id wall thermalboundary 1n	Wall Thermal Boundary 1n	OuterWall	0	degrees	90	degrees	20	m^2	26	
5	34	222	ThermalBoundary	id wall thermalboundary 1w	Wall Thermal Boundary 1w	OuterWall	270	degrees	90	degrees	40	m^2	26	
6	35	222	ThermalBoundary	id wall thermalboundary 1ie	Wall Thermal Boundary 1ie	InteriorWall	90	degrees	90	degrees	40	m^2	26	27
7	36	222	ThermalBoundary	id floor thermalboundary 2	Floor Thermal Boundary 2	BasementFloor	-1	degrees	0	degrees	50	m^2	27	
8	37	222	ThermalBoundary	id ceiling thermalboundary 2	Ceiling Thermal Boundary 2	IntermediaryCeiling	-1	degrees	0	degrees	50	m^2	27	28
9	38	222	ThermalBoundary	id wall thermalboundary 2n	Wall Thermal Boundary 2n	OuterWall	0	degrees	90	degrees	20	m^2	27	
10	39	222	ThermalBoundary	id wall thermalboundary 2s	Wall Thermal Boundary 2s	OuterWall	180	degrees	90	degrees	20	m^2	27	
11	40	222	ThermalBoundary	id wall thermalboundary 2iw	Wall Thermal Boundary 2iw	InteriorWall	270	degrees	90	degrees	40	m^2	27	26
12	41	222	ThermalBoundary	id wall thermalboundary 2e	Wall Thermal Boundary 2e	OuterWall	90	degrees	90	degrees	40	m^2	27	
13	42	222	ThermalBoundary	id floor thermalboundary 31	Floor Thermal Boundary 31	IntermediaryFloor	-1	degrees	0	degrees	50	m^2	28	26
14	43	222	ThermalBoundary	id floor thermalboundary 32	Floor Thermal Boundary 32	IntermediaryFloor	-1	degrees	0	degrees	50	m^2	28	27

Figure 20: Example of the NRG_THERMAL_BOUNDARY view, merging data from tables CITYOBJECT and NRG_THERMAL_BOUNDARY

4 Test data

For testing purposes, and in order to simplify the understanding of the database structure, some test data are also provided.

The test data can be optionally installed at the end of a successful installation of the Energy ADE extension. For nearly all Energy ADE tables (and the associated "vanilla" ones), some data are stored into the tables of the selected 3DCityDB instance.

Please note: the provided test data are meant specifically for testing purposes. The physical meaning of some numeric values (U-values, g-values, time series) might therefore be implausible. Nevertheless, more "realistic" test data will be provided in the upcoming releases.

The test data consist mostly of one building (having ID=1) which is modelled in LoD2. The building has 4 thermal zones (IDs 26 to 29), each of them delimited by a number of thermal boundaries (IDs 30 to 55) and thermal openings (IDs 57 to 7l and 165). A 3D visualisation of the LOD2 geometries and of the 4 thermal zones is given in Figure 22.

Some additional data like Energy Performance Certificates or Refurbishment measures are also provided. For the thermal boundaries and openings, information about multi-layered constructions is also provided.

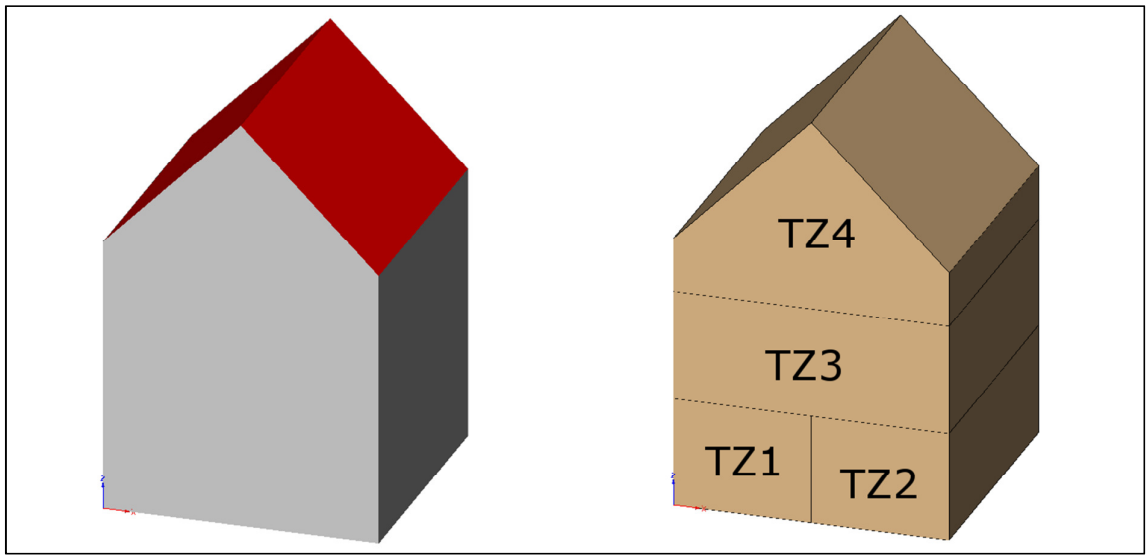


Figure 22: 3D visualisation of the building modelled in the test data. LoD2 geometries (left) and the 4 thermal zones (right).

Each thermal zone is linked to a usage zone (IDs 72 to 75), and each usage zone is characterised by a number of attributes, as well as schedules for heating and cooling. What is more, each usage zone is linked to building units (IDs 76 to 79), facilities (IDs 80 to 91) and occupants. Occupants are also linked to households data and are characterised by occupancy schedules.

A number of energy systems (conversion, distribution, storage) are linked to the above mentioned (city)objects. Both for the whole building and for each thermal zone, energy demand values are provided.

In addition, there are 5 more buildings (IDs 1000 to 1003, 1010) which however do not possess any geometries. They are used as simple “placeholders” and are linked to a number of other (city)objects like weather stations, weather data, further energy systems, etc.

5 Installation

The 3D City Database extension for the Energy ADE 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 Energy ADE** for PostgreSQL can be retrieved here:

https://github.com/gioagu/3dcitydb_ade/tree/master/02_energy_ade

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 Energy ADE, 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.

5.2 Automatic full installation

The automatic full 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.

Please note: the automatic full installation process requires that:

a) The 3DCityDB instance is in its “vanilla” state, i.e. no other add-ons, packages, ADEs have been previously installed **AND**
b) The 3DCityDB instance is empty, i.e. no data have been previously loaded (empty database!) **AND**
c) The user wants to install everything (Utilities Package, Metadata Module, Energy ADE “base”, and Energy ADE Updatable views)

If *all* three conditions are met, then this is the recommended installation procedure, otherwise please refer to the next section 5.3 (Manual installation). 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

In the subfolder *install_scripts*, the batch file Energy_ADE_Automatic_Install.bat 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. A simple text editor will suffice. An example of the Energy_ADE_Automatic_Install.bat file is shown in Figure 23. Upon successful installation, the message shown in Figure 24 will be output.

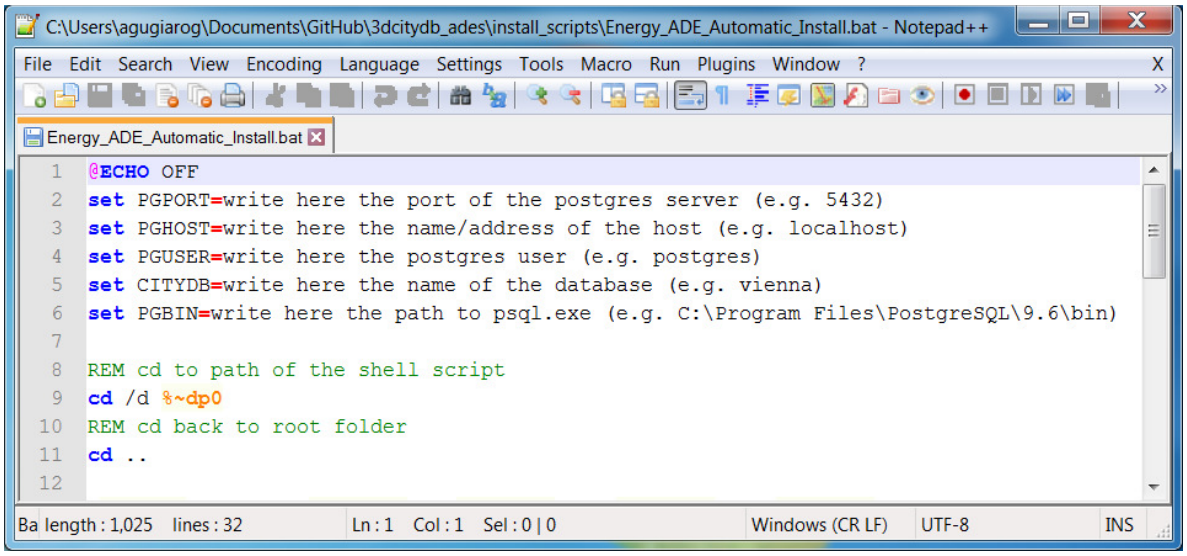


Figure 23: Example of the Energy_ADE_Automatic_Install.bat batch file

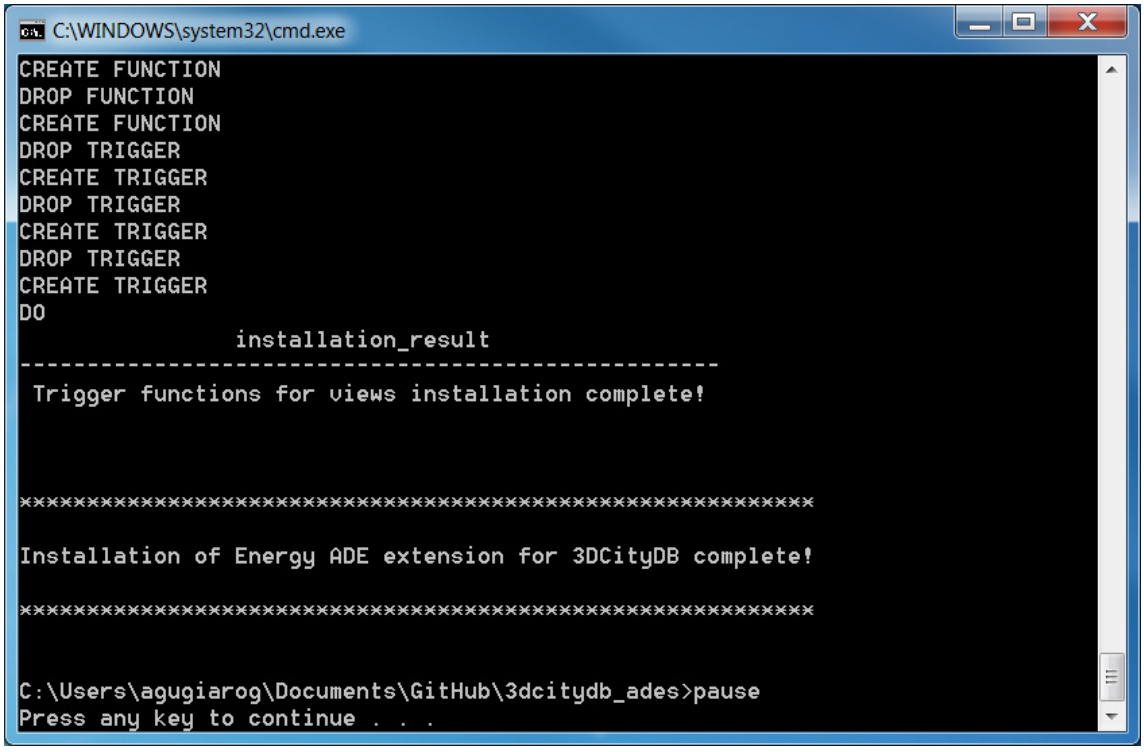


Figure 24: Message upon successful installation

5.3 Manual installation

If at least one of the conditions required for the automatic full installation are not met, or the user prefers not to use the automatic installation scripts, then the user is advised to proceed with the manual approach. This covers in general the situation when the Metadata module, or another ADE or the (optional) Utilities Package have been previously installed. In any case, the steps described in the following require to start from an empty 3DCityDB database instance. Unlike the previous section about the automatic installation process, here some details are added regarding each single installation step.

5.3.1 Installing the Metadata module

Before installing the actual 3DCityDB extension for the Energy ADE itself, some stored procedures and the tables of the Metadata module need to be installed (in case they were not previously installed for another ADE). In order to install the Metadata module, the user needs to perform the steps described in the following.

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 – Execute the SQL scripts in folder *\01_metadata_module\postgresql*

In the folder *\01_metadata_module\postgresql*, the SQL scripts:

01_metadata_module_TABLES.sql,
02_metadata_module_FUNCTIONS.sql,
03_metadata_module_TABLE_DATA.sql

must simply launched **sequentially** using for example the PgAdmin GUI, which is generally installed together with any recent PostgreSQL installation package. See for example Figure 25, where PgAdmin III v. 1.22 is being used. Upon successful installation, messages like the one shown in Figure 26Figure 30 will be output.

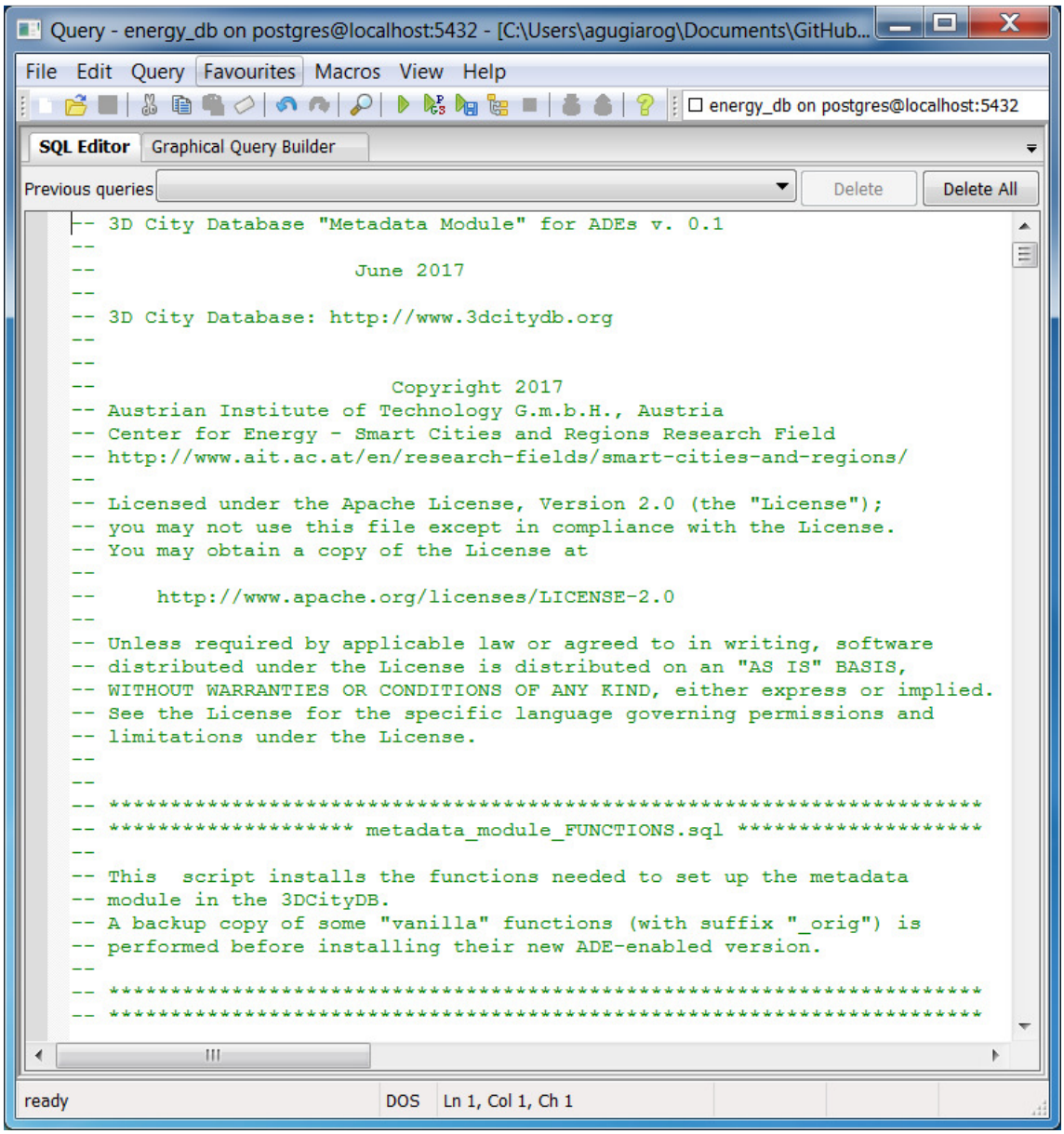


Figure 25: Installation of the Metadata module using PgAdmin III

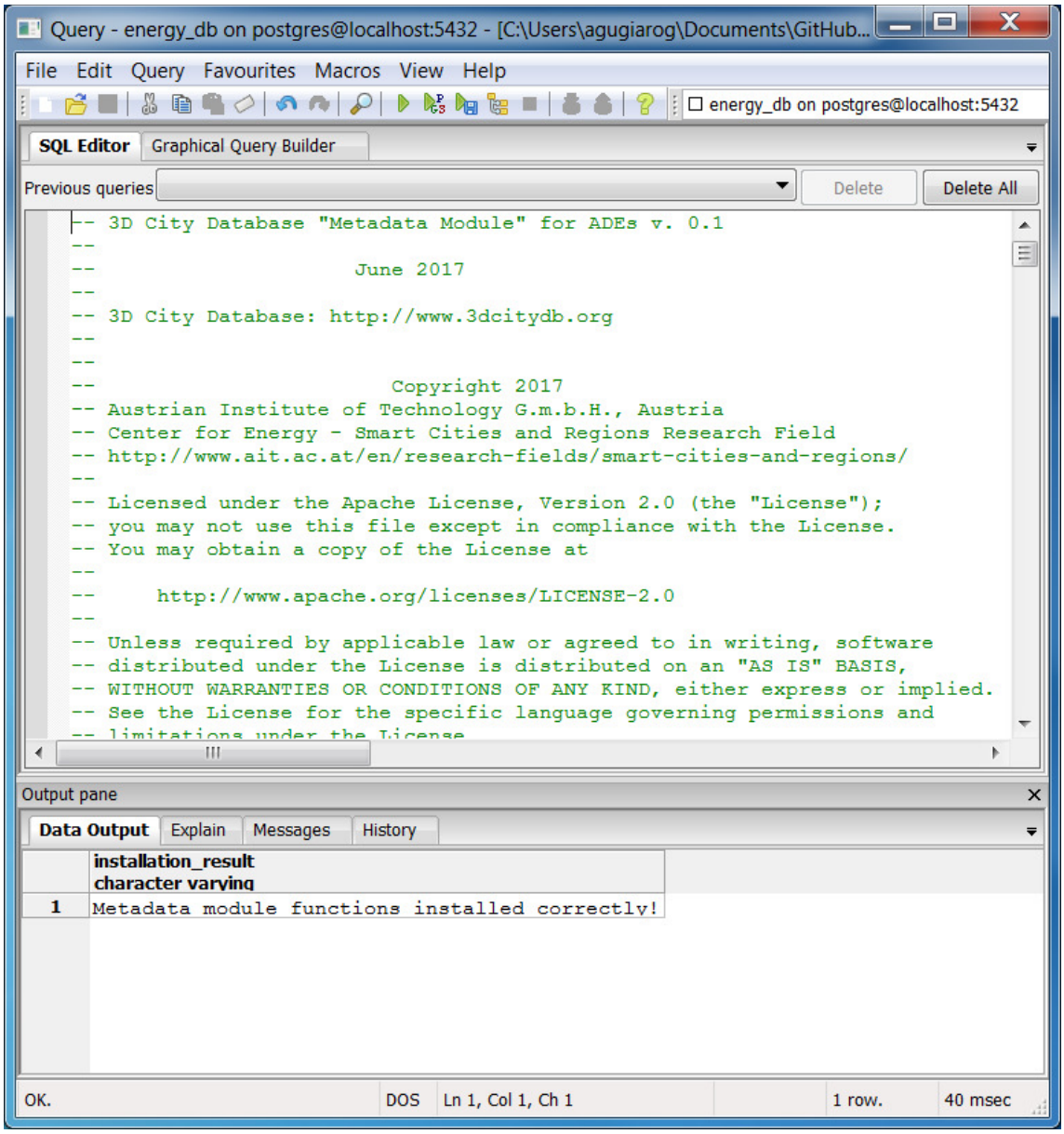
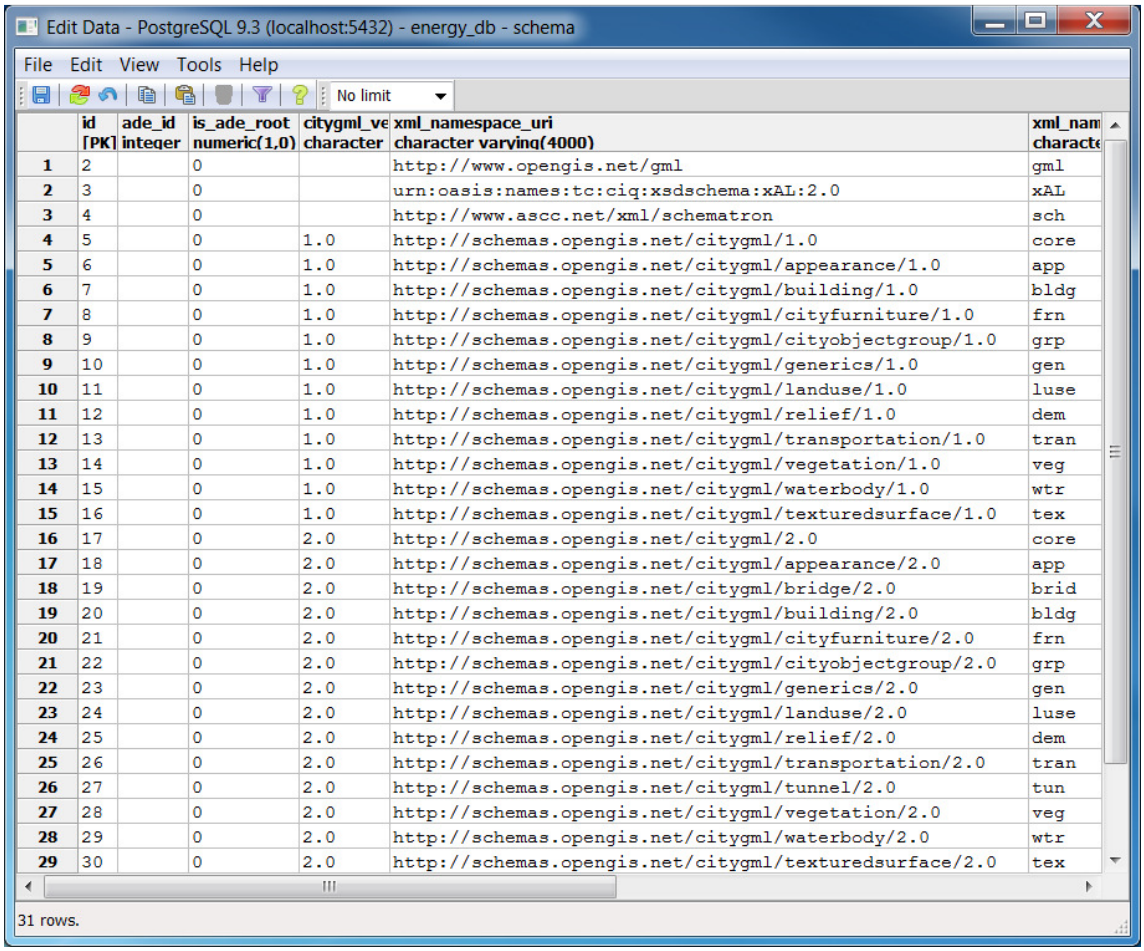


Figure 26: Installation of the SQL script from within the PgAdmin III GUI. Upon successful installation, the resulting message can be read in the lower part of the window (Output pane)

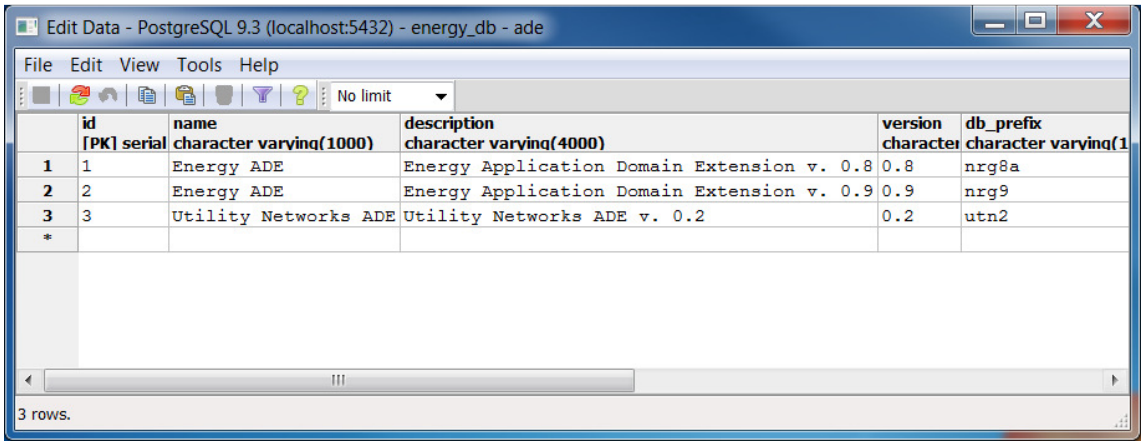
Upon installation of the Metadata module, the ADE table will be empty, however some data will be added for example to table SCHEMA (and to tables SCHEMA_REFERENCING, OBJECTCLASS, and SCHEMA_TO_OBJECTCLASS, too). An example of table SCHEMA is shown in Figure 27.



	id [PK] integer	ade_id integer	is_ade_root numeric(1,0)	citygml_ve character	xml_namespace_uri character varying(4000)	xml_nam characte
1	2		0		http://www.opengis.net/gml	gml
2	3		0		urn:oasis:names:tc:ciq:xsdschema:xAL:2.0	xAL
3	4		0		http://www.ascc.net/xml/schematron	sch
4	5		0	1.0	http://schemas.opengis.net/citygml/1.0	core
5	6		0	1.0	http://schemas.opengis.net/citygml/appearance/1.0	app
6	7		0	1.0	http://schemas.opengis.net/citygml/building/1.0	bldg
7	8		0	1.0	http://schemas.opengis.net/citygml/cityfurniture/1.0	frn
8	9		0	1.0	http://schemas.opengis.net/citygml/cityobjectgroup/1.0	grp
9	10		0	1.0	http://schemas.opengis.net/citygml/generics/1.0	gen
10	11		0	1.0	http://schemas.opengis.net/citygml/landuse/1.0	luse
11	12		0	1.0	http://schemas.opengis.net/citygml/relief/1.0	dem
12	13		0	1.0	http://schemas.opengis.net/citygml/transportation/1.0	tran
13	14		0	1.0	http://schemas.opengis.net/citygml/vegetation/1.0	veg
14	15		0	1.0	http://schemas.opengis.net/citygml/waterbody/1.0	wtr
15	16		0	1.0	http://schemas.opengis.net/citygml/texturedsurface/1.0	tex
16	17		0	2.0	http://schemas.opengis.net/citygml/2.0	core
17	18		0	2.0	http://schemas.opengis.net/citygml/appearance/2.0	app
18	19		0	2.0	http://schemas.opengis.net/citygml/bridge/2.0	brid
19	20		0	2.0	http://schemas.opengis.net/citygml/building/2.0	bldg
20	21		0	2.0	http://schemas.opengis.net/citygml/cityfurniture/2.0	frn
21	22		0	2.0	http://schemas.opengis.net/citygml/cityobjectgroup/2.0	grp
22	23		0	2.0	http://schemas.opengis.net/citygml/generics/2.0	gen
23	24		0	2.0	http://schemas.opengis.net/citygml/landuse/2.0	luse
24	25		0	2.0	http://schemas.opengis.net/citygml/relief/2.0	dem
25	26		0	2.0	http://schemas.opengis.net/citygml/transportation/2.0	tran
26	27		0	2.0	http://schemas.opengis.net/citygml/tunnel/2.0	tun
27	28		0	2.0	http://schemas.opengis.net/citygml/vegetation/2.0	veg
28	29		0	2.0	http://schemas.opengis.net/citygml/waterbody/2.0	wtr
29	30		0	2.0	http://schemas.opengis.net/citygml/texturedsurface/2.0	tex

Figure 27: Example of the SCHEMA table after installing the Metadata module for the first time.

Whenever installing a new ADE, a new entry will be added to the ADE table. Figure 28 shows for example that different ADEs, and different versions of the same ADE, can be installed concurrently, as long the values in the db_prefix column remain unique.



	id [PK] serial	name character varying(1000)	description character varying(4000)	version character	db_prefix character varying(1
1	1	Energy ADE	Energy Application Domain Extension v. 0.8	0.8	nrg8a
2	2	Energy ADE	Energy Application Domain Extension v. 0.9	0.9	nrg9
3	3	Utility Networks ADE	Utility Networks ADE v. 0.2	0.2	utn2
*					

Figure 28: Example of table ADE showing (some) information about installed ADEs

During the installation of the Metadata module, the only "vanilla" 3DCityDB table that is changed is the OBJECTCLASS one, which is extended by adding some columns. Please refer to section 3.1.1 for more details.

In addition, the vanilla stored procedures:

```
citydb_pkg.delete_cityobject(…),
citydb_pkg.intern_delete_cityobject(…),
citydb_pkg.get_envelope_cityobject(…),
citydb_pkg.objectclass_id_to_table_name(…),
citydb_pkg.cleanup_schema(…)
```

are first backed up by renaming them with a *"_orig*" suffix. Successively, new ADE-enabled versions of these stored procedures are installed.

Some additional stored procedures are also installed in the *citydb_pkg* schema and are meant to offer a set of stored procedures to deal with ADEs. They are, for example:

```
citydb_pkg.get_objectclass_info(…),
citydb_pkg.get_classname(…),
citydb_pkg.objectclass_id_to_table_name(…),
citydb_pkg.objectclass_classname_to_id(…),
citydb_pkg.change_ade_column_srid(…),
citydb_pkg.drop_ade(…)
```

and the related

```
citydb_pkg.drop_ade_objectclasses(…),
citydb_pkg.drop_ade_tables(…),
citydb_pkg.drop_ade_functions(…).
```

Most notably, function `drop_ade(…)` allows to remove all objects created by a specific ADE. The stored procedure requires as input only the ID of the installed ADE from the ADE table. Taking the last entry of the table in Figure 28 as example (e.g. "Utility Networks ADE v. 0.2"), to remove this ADE, the stored procedure needs to be executed as follows:

```
SELECT citydb_pkg.drop_ade(3);
```

As a result, all objects created by this ADE (tables, stored procedures, views, constraints, etc.) will be dropped, while all "vanilla" objects of the 3DCityDB (and the Metadata module) will be kept.

(Alternative) Step 2 – Execute the batch file CREATE_DB_Metadata_module.bat

Instead of manually launching the SQL scripts, the batch file 00_CREATE_DB_Metadata_module.bat in folder *\01_metadata_module* can be run from a Windows command shell (or simply by double-clicking it), after the configuration parameters regarding the PostgreSQL server and the selected database have been adapted – in a similar way as already described for the automatic full installation. An example of the CREATE_DB_Metadata_module.bat file is shown in Figure 29. For Linux environments, the

equivalent file CREATE_DB_Metadata_module.sh is also provided. Upon successful installation, the message shown in Figure 30 will be output.

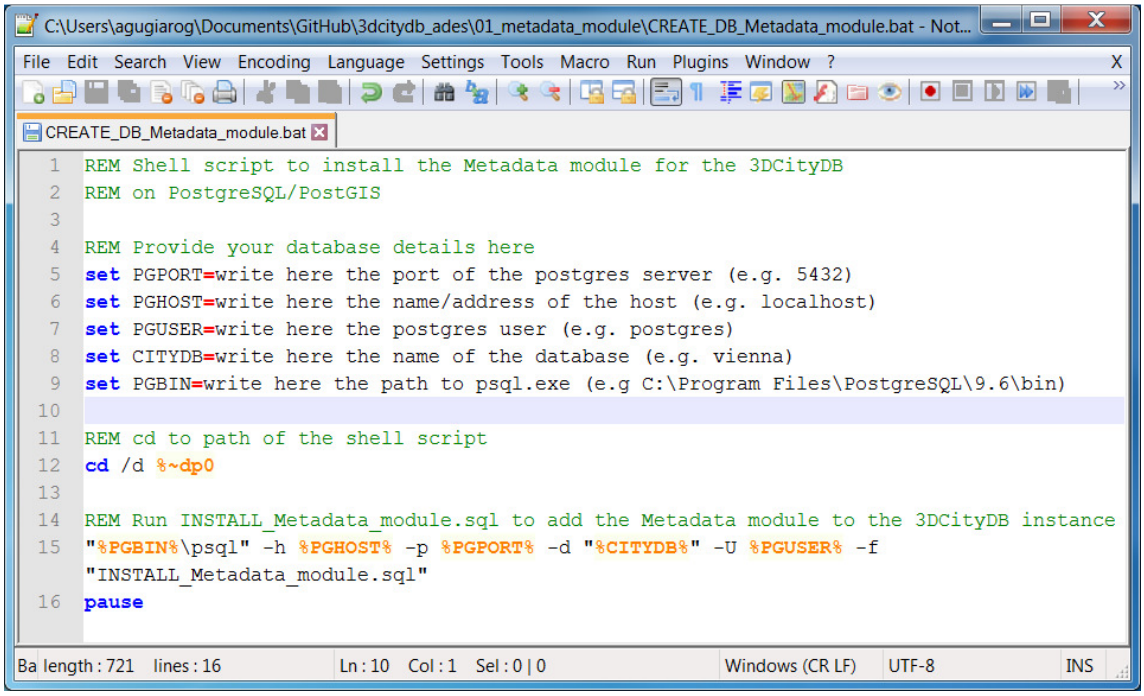


Figure 29: Example of the 00_CREATE_DB_Metadata_module.bat batch file

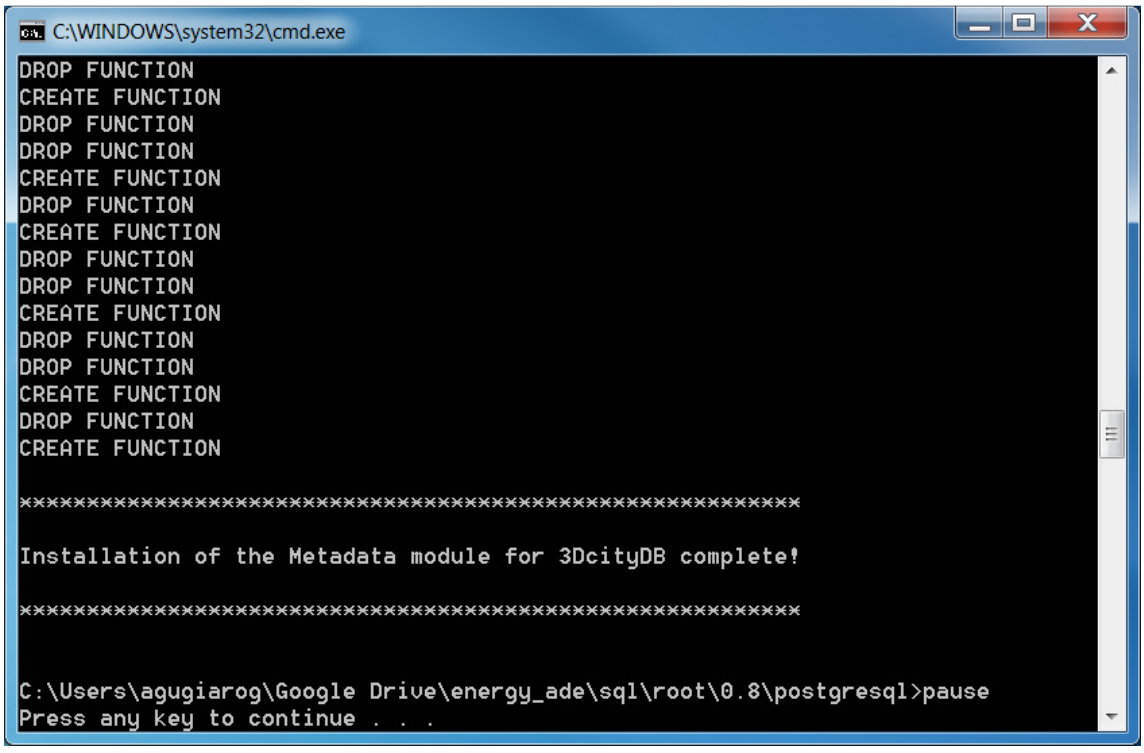


Figure 30: Message upon successful installation of the Metadata module for the 3DCityDB

5.3.2 Installing 3DCityDB extension for the Energy ADE

The installation process of the 3DCityDB extension for the Energy ADE is analogous to the one described in the previous section. Please remember that *in any case* a successful installation of the Metadata module is a prerequisite for a successful installation of the 3DCityDB extension for the Energy ADE. The Energy ADE comes with a "base" and an "optional" installation part. With regards to the SQL scripts listed in the following, the "optional" part consists of the views and all related stored procedures to make them updatable (i.e. from script 06_Energy_ADE_VIEWS.sql onward). If the user wants or needs also the "optional" part, then the 3DCityDB Utilities Package (as described in Appendix A) needs to be installed first. Please refer to the Appendix A for further information about installing the 3DCityDB Utilities Package.

In any case, in order to install the 3DCityDB extension for the Energy ADE, 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 *\02_energy_ade\postgresql*

As already explained for the Metadata module, in the folder ***\02_energy_ade\postgresql*** the SQL scripts

01_Energy_ADE_FUNCTIONS.sql,
02_Energy_ADE_DML_FUNCTIONS.sql,
03_Energy_ADE_TABLES.sql,
04_Energy_ADE_TRIGGERS.sql,
05_Energy_ADE_TABLE_DATA.sql,
06_Energy_ADE_VIEWS.sql, (requires 3DCityDB Utilities Package)
07_Energy_ADE_VIEW_FUNCTION.sql, (requires 3DCityDB Utilities Package)
08_Energy_ADE_VIEW_TRIGGERS.sql, (requires 3DCityDB Utilities Package)

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

Upon installation, the corresponding Energy ADE entry will be added to ADE table, as shown in Figure 28. All new tables will be added to the *citydb* schema, all new stored procedures will be installed in the *citydb_pkg* schema, and (optionally) all updatable views will be installed in the newly added *citydb_view* schema. All objects have names prefixed with the db_prefix value.

During the installation process, the stored procedure `citydb_pkg.change_ade_column_srid(…)` is automatically executed in order to set the SRID of the newly added geometry columns which come with the new tables. The SRID is the same one used throughout the 3DCityDB instance: the information is read from the DATABASE_SRS table.

(Alternative) Step 2 – Execute the batch file CREATE_DB_Energy_ADE.bat

As already explained for the Metadata module, the batch file CREATE_DB_Energy_ADE.bat can be run from a Windows command shell (or simply by double-clicking it). However, also in this case the configuration parameters regarding the PostgreSQL server and the selected database must be first adapted accordingly. For Linux environments, the equivalent file CREATE_DB_Energy_ADE.sh is also provided.

Please note: The batch file install both the "base" and the "optional" parts. The user must therefore be sure to have previously installed the 3DCityDB Utilities Package.

Upon successful installation, the message shown in Figure 31 will be output.

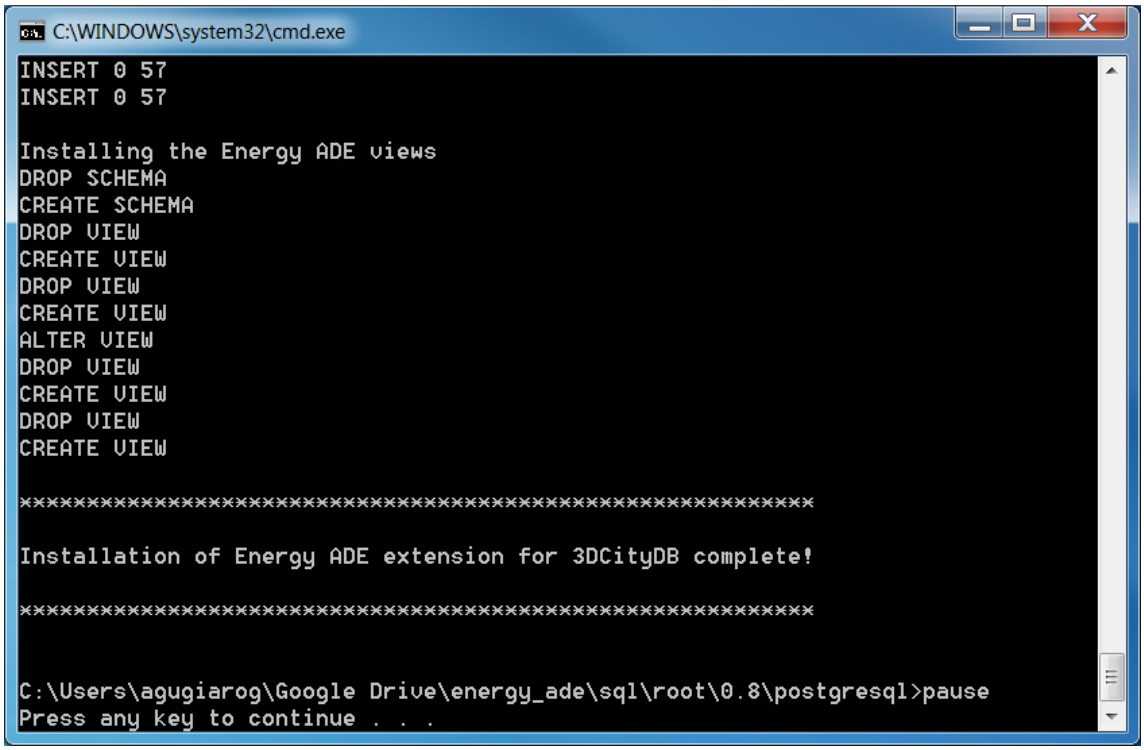


Figure 31: Message upon successful installation of the Energy ADE extension for the 3DCityDB

5.4 Installation of the test data

In order to install the test data, an instance of the 3DCityDB using the **EPSG code 31256** is needed. In addition, both the Metadata module and the 3DCityDB extension for the Energy ADE need to be installed, as described in the previous sections.

All test data are contained within on single SQL script file, named Energy_ADE_TEST_DATA.sql and located in the \02_energy_ade\test_data folder.

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

Appendix A: The 3DCityDB Utilities Package

Please note: this Appendix is just a slightly edited copy of the documentation shipped with the 3DCityDB Utilities Package as of August 2017. It is added to this manual just for the sake of completeness. However, the user is encouraged to check for the most recent and updated version at https://github.com/gioagu/3dcitydb_ade/tree/master/00_utilities_package/manual

The 3D City Database Utilities Package contains a number of additional stored procedures, views and trigger functions which extend the "vanilla" version of the 3DCityDB. The Utilities Package is intended to be installed optionally on top of a 3DCityDB database instance, and the overall goal is to offer additional features and functionalities which help in the data management tasks.

As a matter of fact, data handling within the 3DCityDB can be sometimes not immediately straightforward due to the complexity of the database schema (which indeed reflects the richness and complexity of the CityGML data model). For example, data belonging to a specific class can be stored in different linked tables in the database. Therefore CRUD[5] operations (i.e. select, insert, update and delete) may result in complex SQL statements. In order to partially overcome this complexity, the 3DCityDB is already shipped with a number of stored procedures, e.g. the "delete" ones. These are of particular help as the complexity of the ordered, hierarchical deletion process is carried out automatically and hidden from the user.

A.1 Content

In order to further facilitate CRUD operations at database level, the Utility Package provides in addition:

- Insert stored procedures for some of the "vanilla" 3DCityDB tables;
- Views offering a simplified access to data spread over multiple tables. All views are created in a separated database schema named *citydb_view*;
- Additional "smart" CRUD stored procedures (for insert and delete operations) in the *citydb_view* schema;
- Trigger functions built upon the aforementioned "smart" stored procedures which make most of the views updatable. As a consequence, the user can interact directly with the views as if they were normal tables and perform not only select operations, but also insert, update and delete.

A.1.1 Insert stored procedures in schema *citydb_pkg*

The insert stored procedures allow to insert data into the table they refer to. They are stored in the *citydb_pkg* schema and are named using the table name prefixed with "insert_", e.g. `citydb_pkg.insert_building(…)` or `citydb_pkg.insert_cityobject(…)`.

[5] CRUD represents an acronym for the database operations **C**reate, **R**ead, **U**pdate, and **D**elete.

All stored procedures are written in PL/pgSQL and they all have parameters that are called using the named notation[6]. The named notation is especially useful for stored procedures that have a large number of parameters, since it makes the associations between parameters and actual arguments more explicit and reliable. In the named notation, the arguments are matched to the stored procedures' parameters by name and can be written in any order. Parameters that have default values given in the stored procedure's declaration need not to be written at all in the call. This is particularly useful since any combination of parameters can be omitted. In order to use an insert stored procedure with the named notation, each argument's name is specified using **:=** to separate it from the argument expression. For example:

```
SELECT citydb_pkg.insert_building(
id := 5094,
building_root_id := 5094,
class := 'Residential',
storeys_above_ground := 5,
storeys_below_ground := 2
);
```

The insert stored procedures have the following characteristics:

- All parameters are defined as DEFAULT NULL, i.e. they are all optional, except for:
  - The OBJECTCLASS_ID parameter, if the table has such attribute (e.g. table CITYOBJECT);
  - The ID parameter, if the table has a primary key which is *not* generated by a sequence (e.g. tables BUILDING);
  - Both parameters forming the primary key of an association table (e.g. GROUP_TO_CITYOBJECT). Such association tables can be easily recognised by the "_TO_" string in the table name;
- For tables having the ID generated by a sequence, there are two possibilities:
  - The user does not specify the ID parameter. In this case the next available value is retrieved from the corresponding sequence automatically and assigned to the new record to be inserted;
  - The user specifies the ID parameter. In this case, it is the responsibility of the user to choose the ID value properly, in order to comply with the UNIQUE constraint in the corresponding table. If the ID value is already used, an error will be raised and the insert operation aborted.
- If the GMLID parameter is not given by the user, then a universally unique identifier (UUID) is generated automatically and assigned to the new record to be inserted;
- Upon successful completion, the insert stored procedures return the ID of the inserted record. In case of an association table, the stored procedure returns null.
- In case of an EXCEPTION, a notice is raised with the error message.

[6] The other approach is called *positional notation*. For more details, please refer to https://www.postgresql.org/docs/9.1/static/sql-syntax-calling-funcs.html

A.1.2 Views

As mentioned before, a number of views is installed in the database schema *citydb_view*. Regarding the **naming convention**, most of the views allow to access data for specific object classes. The name of the view coincides or approximates the name of the corresponding class in the OBJECTCLASS table. For example, view BUILDING presents data of Building objects, while view BUILDING_PART presents data of BuildingPart objects. In addition, wherever appropriate, not only the OBJECTCLASS_ID attribute, but also the **CLASSNAME** one (retrieved from the joined OBJECTCLASS table) are included in the view.

Please note: in order to access *any* database object in the *citydb_view* schema, **qualified names**[7] consisting of the schema name and the object name separated by a dot ***must* be used**. This is a design decision, in order to avoid homonymy issues since some objects (e.g. views) have the same names of other objects in the *citydb* and *citydb_pkg* schemas. 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 5 lists the views shipped with this version of the 3DCityDB Utilities Package, while Figure 32 presents an example of view BUILDING.

VIEW	UPDATABLE?
building	✓
building_furniture	
building_installation	
building_installation_exterior	
building_installation_interior	
building_lod0_footprint	
building_part	✓
cityobjectgroup	✓
generic_cityobject	
group_to_cityobject	
land_use	
opening	
opening_door	
opening_window	
plant_cover	
room	
solitary_vegetat_object	
thematic_surface	
thematic_surface_ceiling	
thematic_surface_closure	
thematic_surface_floor	
thematic_surface_ground	
thematic_surface_interior_wall	
thematic_surface_outer_ceiling	
thematic_surface_outer_floor	

[7] For more details, see https://www.postgresql.org/docs/9.1/static/ddl-schemas.html

VIEW	UPDATABLE?
thematic_surface_roof	
thematic_surface_wall	
waterbody	

Table 5: Views shipped with the current version of the Utilitites Package. All views are stored in schema *citydb_view*.

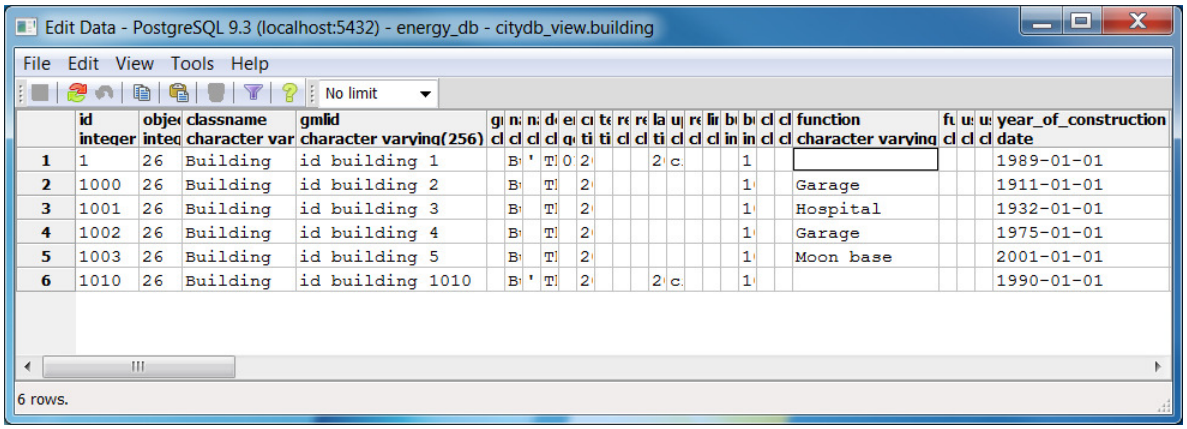


Figure 32: Example of view BUILDING

A.1.3 Additional stored procedures in schema citydb_view

Within the schema *citydb_view* a number of stored procedures is installed. They can be grouped in two groups: delete and insert stored procedures.

The **delete stored procedures** work conceptually like the homologous ones in the *citydb_pkg* schema. They simplify the deletion process, especially in case of complex objects stored in multiple tables. Moreover, they are used by the trigger functions (see later).

The "smart" **insert stored procedures** build upon the aforementioned ones in the *citydb_pkg* schema, however with a number of differences, namely:

a) They allow to insert in one statement all attributes of an object which may be spread over multiple tables. For example, the stored procedure `citydb_view.insert_building(…)` takes as input *all* parameters of the underlying tables CITYOBJECT *and* BUILDING, and takes care of inserting the data appropriately. In this example, data of table CITYOBJECT is first inserted, then data of table BUILDING;
b) The user *must not* provide the OBJECTCLASS_ID value as input parameter of the stored procedure. If required, each stored procedures looks up automatically for the proper OBJECTCLASS_ID in table OBJECTCLASS and uses it. For example, `citydb_view.insert_building(…)` automatically retrieves the value 26 and uses it in the following insert statements;
c) For the input of the ID parameters, the approach is the same like in the insert stored procedures in schema *citydb_pkg*: the user can leave it blank (thus automatically getting the next available value from the corresponding sequence), or provide a value. In either case, the ID is used and passed to all tables involved in the cascading insert operation.

d) Upon successful completion, the insert stored procedures return the ID of the inserted record. In case of an association table, the stored procedure returns null.
e) In case of an EXCEPTION, a notice is raised with the error message.

A.1.4 Trigger functions

For nearly all views, a number of triggers and trigger functions are added, in order to make the views updatable. Therefore, if a view is also updatable, entries can be updated, inserted and deleted directly interacting with the view. Whether a view is updatable or not is shown in Table 5. Please note that, by design decision, the underlying update trigger functions do *not* allow to change neither the ID nor the OBJECTCLASS_ID attributes.

Although still limited at the time of writing, the number of database object shipped with the 3DCityDB Utilities Package will increase in the upcoming releases of the Utilities Package. It is nevertheless important to remark that the same concepts will apply for database objects shipped with ADEs (views, stored procedures, triggers, etc.), which will build upon some of the objects shipped with this package.

A.2 Installation

The source code and the documentation the 3DCityDB **Utilities Package** for PostgreSQL can be retrieved here: https://github.com/gioagu/3dcitydb_ade/tree/master/00_utilities_package

A.2.1 System requirements

In order to set up the 3DCityDB Utilities 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. PostGIS 2.0 or higher is required.

A.2.2 Automatic installation

The (recommended) installation process can be carried out nearly completely automatically. The user simply needs to run from a Windows command shell (or simply by double-clicking it) the batch file CREATE_DB_citydb_utilities.bat. However, the configuration parameters regarding the PostgreSQL server and the selected database must be first adapted. A simple text editor will suffice. Please note that the installation should conveniently be carried out using the same user that installed the *citydb* schema.

An example of the CREATE_DB_citydb_utilities.bat file is shown in Figure 33. For Linux environments, the equivalent file CREATE_DB_citydb_utilities.sh is also provided. Upon successful installation, the message shown in Figure 34 will be output.

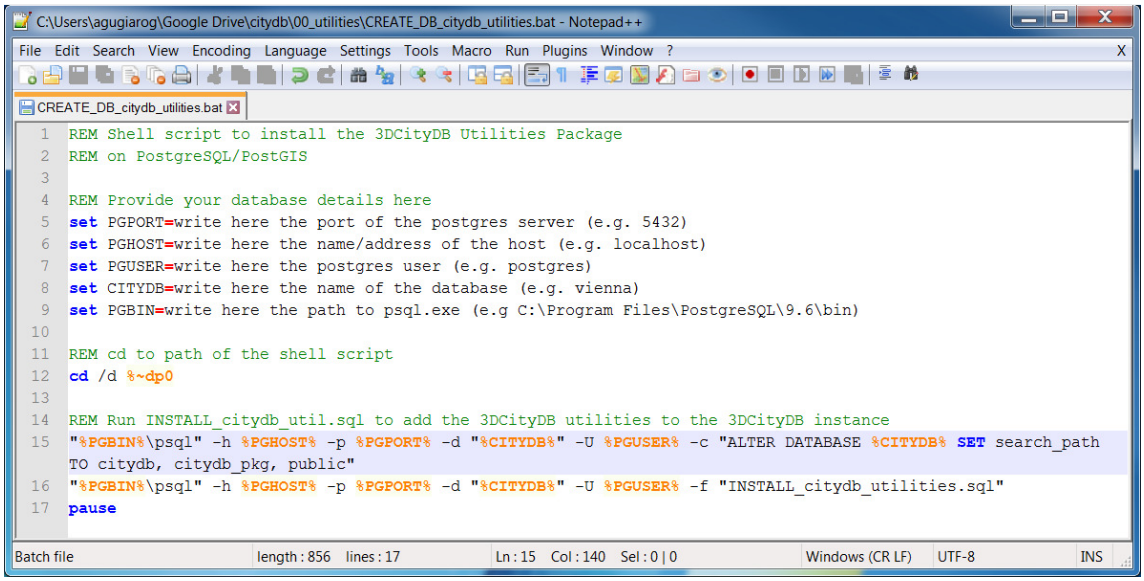


Figure 33: Example of the CREATE_DB_citydb_utilities.bat batch file

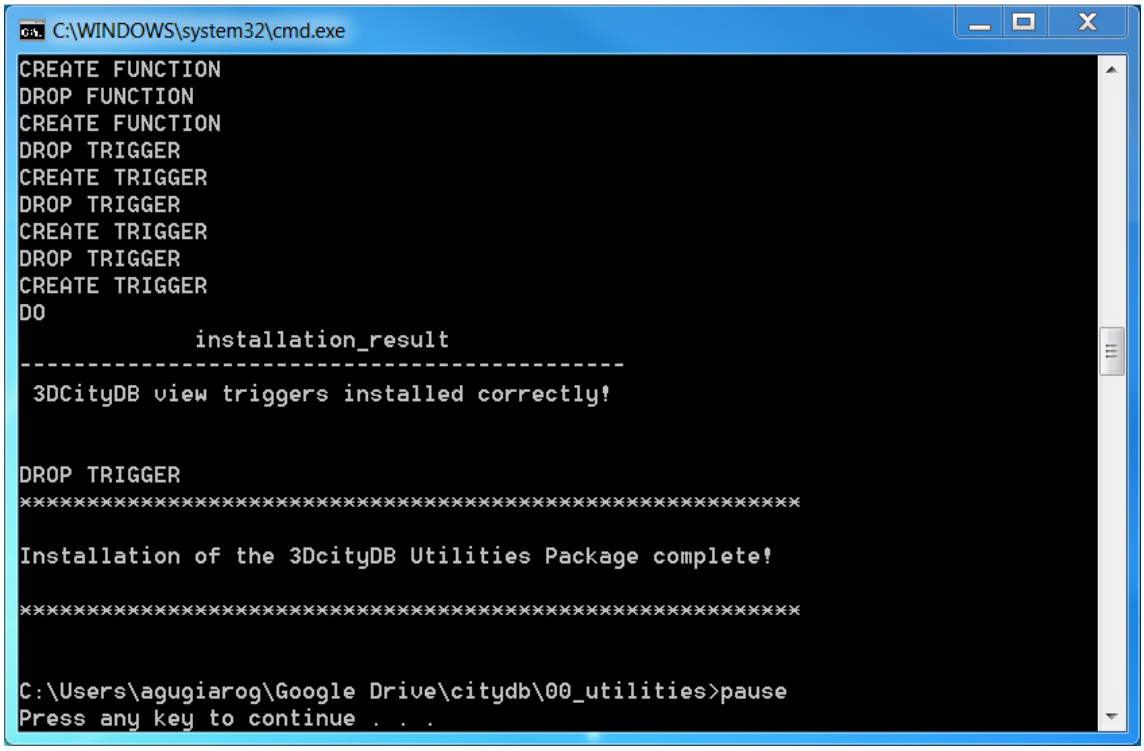


Figure 34: Message upon successful installation of the 3DCityDB Utilities Package

A.2.3 (Alternative) manual installation

As an alternative, manual, installation process, the user must **sequentially** launch the following SQL scripts, using for example the PgAdmin GUI, which is generally installed together with any recent PostgreSQL installation package. See for example Figure 35, where PgAdmin III v. 1.22 is being used. Upon successful installation, messages like the one shown in Figure 36Figure 30 will be output.

The SQL scripts can be found in the \00_utilities\postgresql subfolder and are:

01_citydb_util_DML_FUNCTIONS.sql,
02_citydb_util_VIEWS.sql,
03_citydb_util_VIEW_FUNCTIONS.sql,
04_citydb_util_VIEW_TRIGGERS.sql.

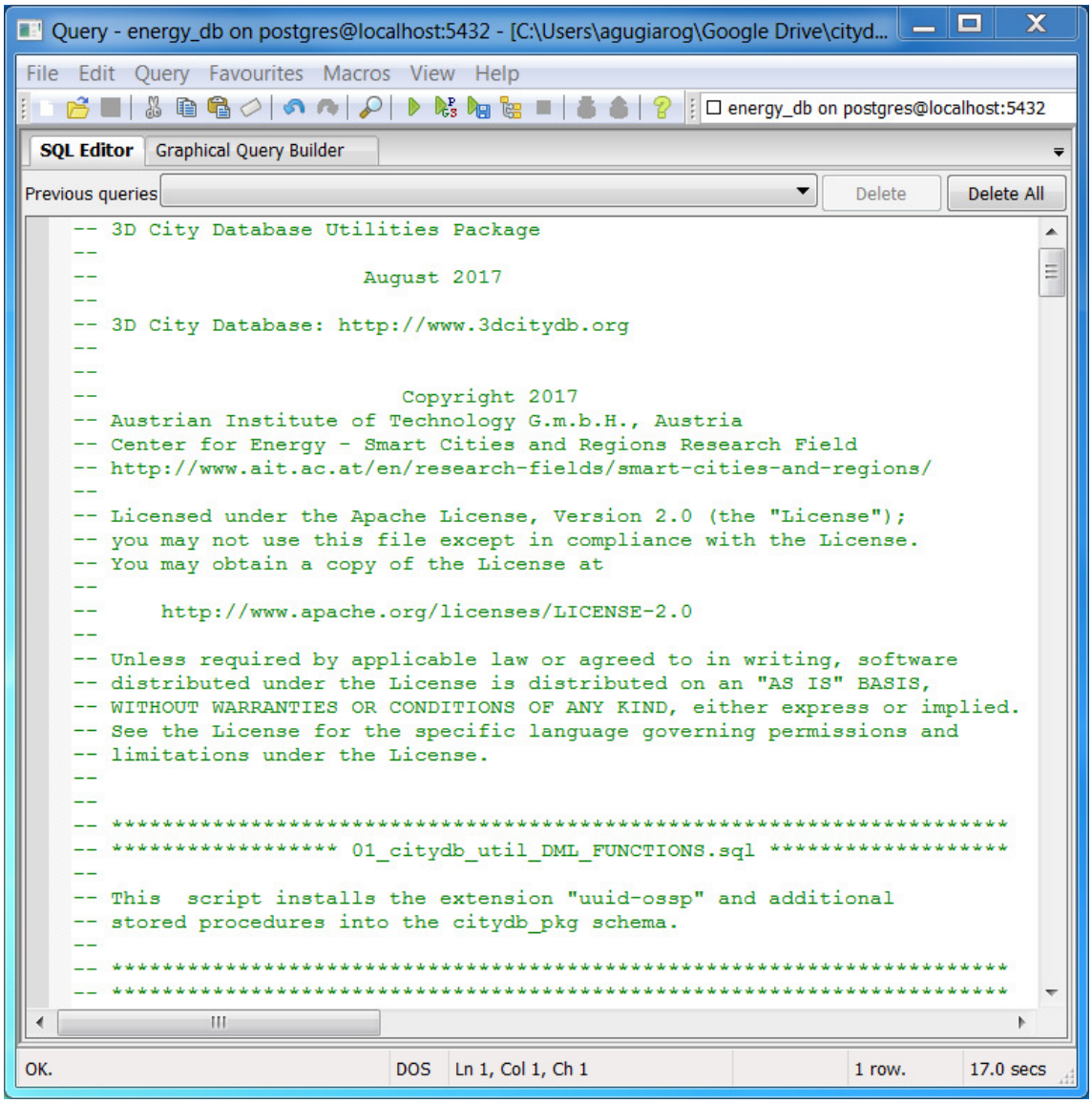


Figure 35: Installation of the Utilities Package using PgAdmin III

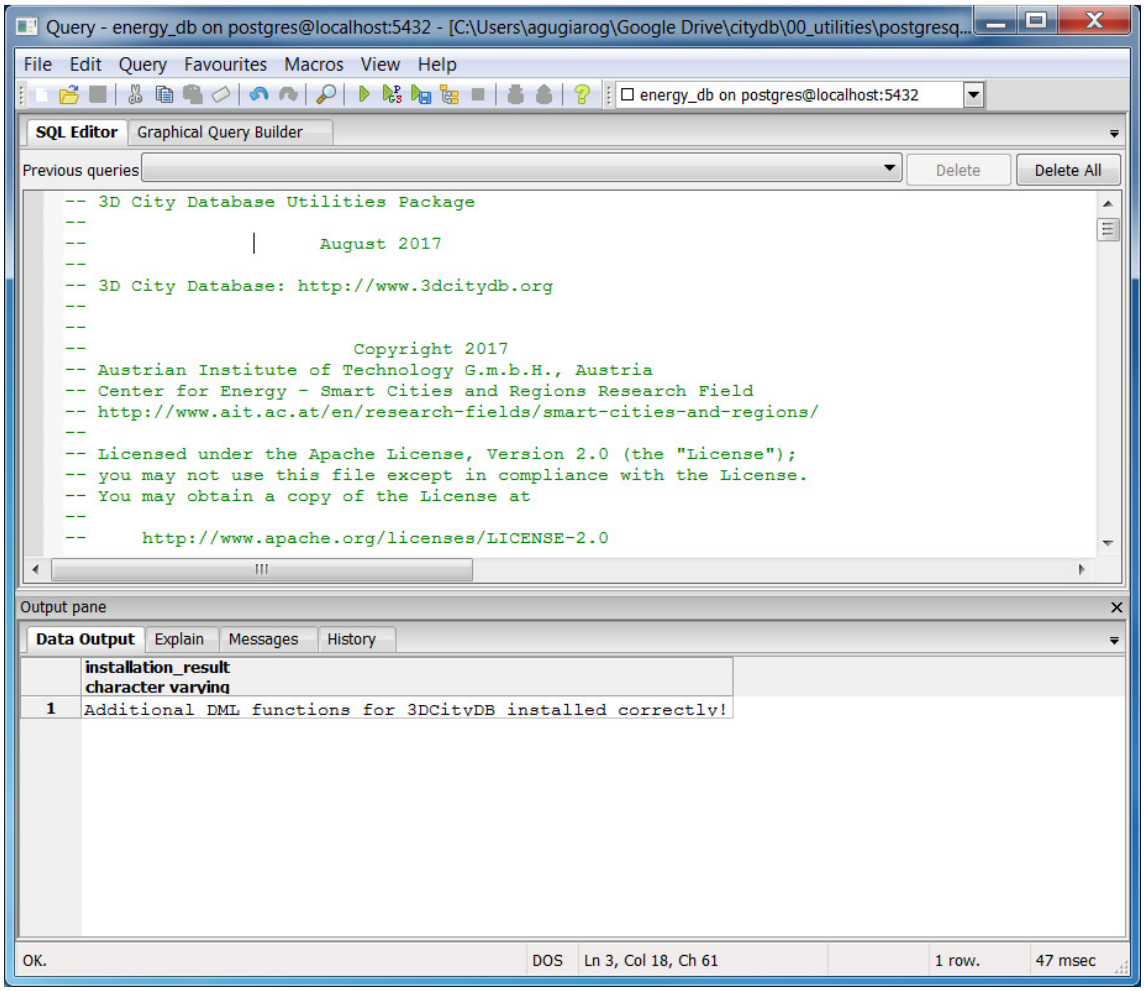
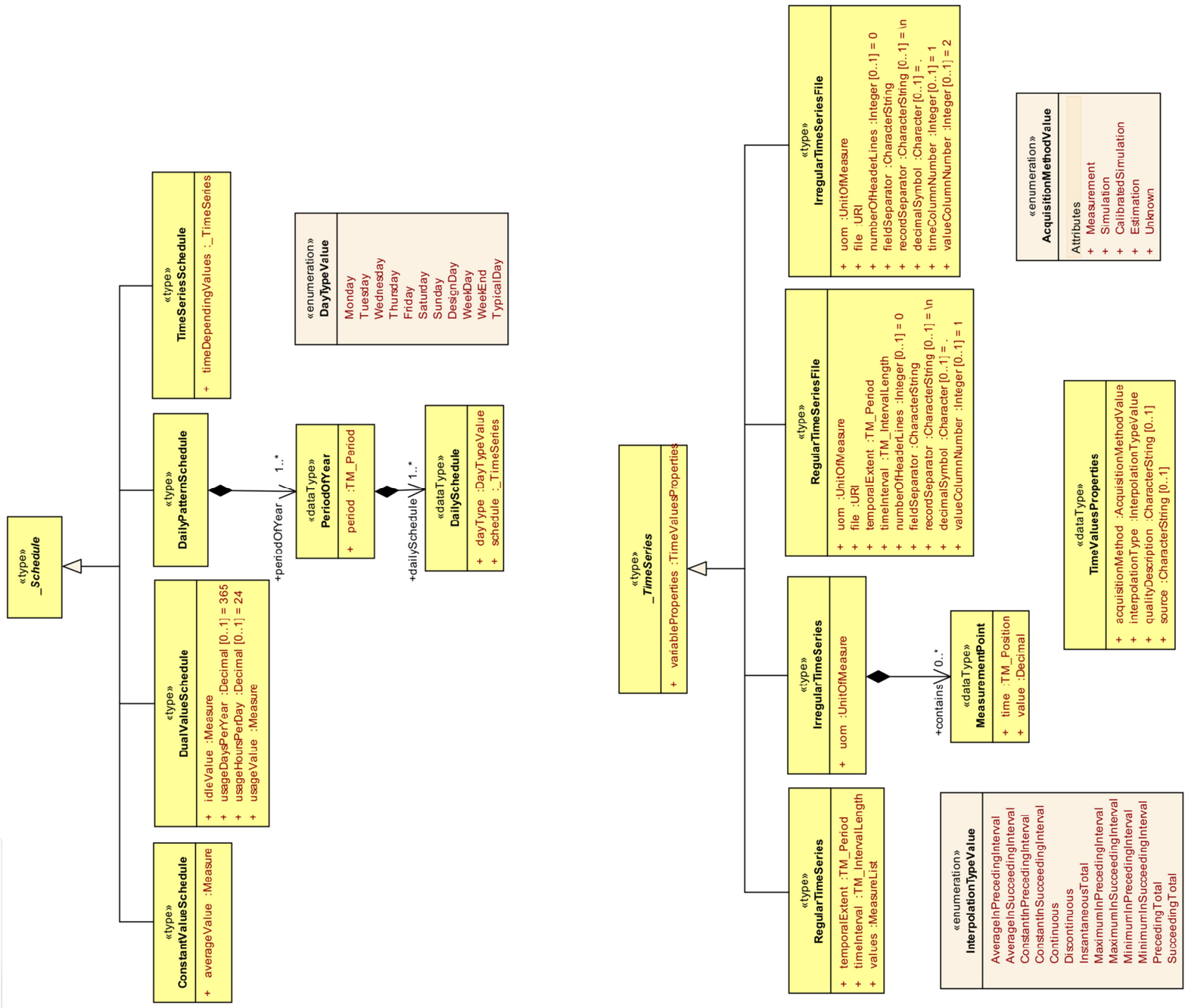
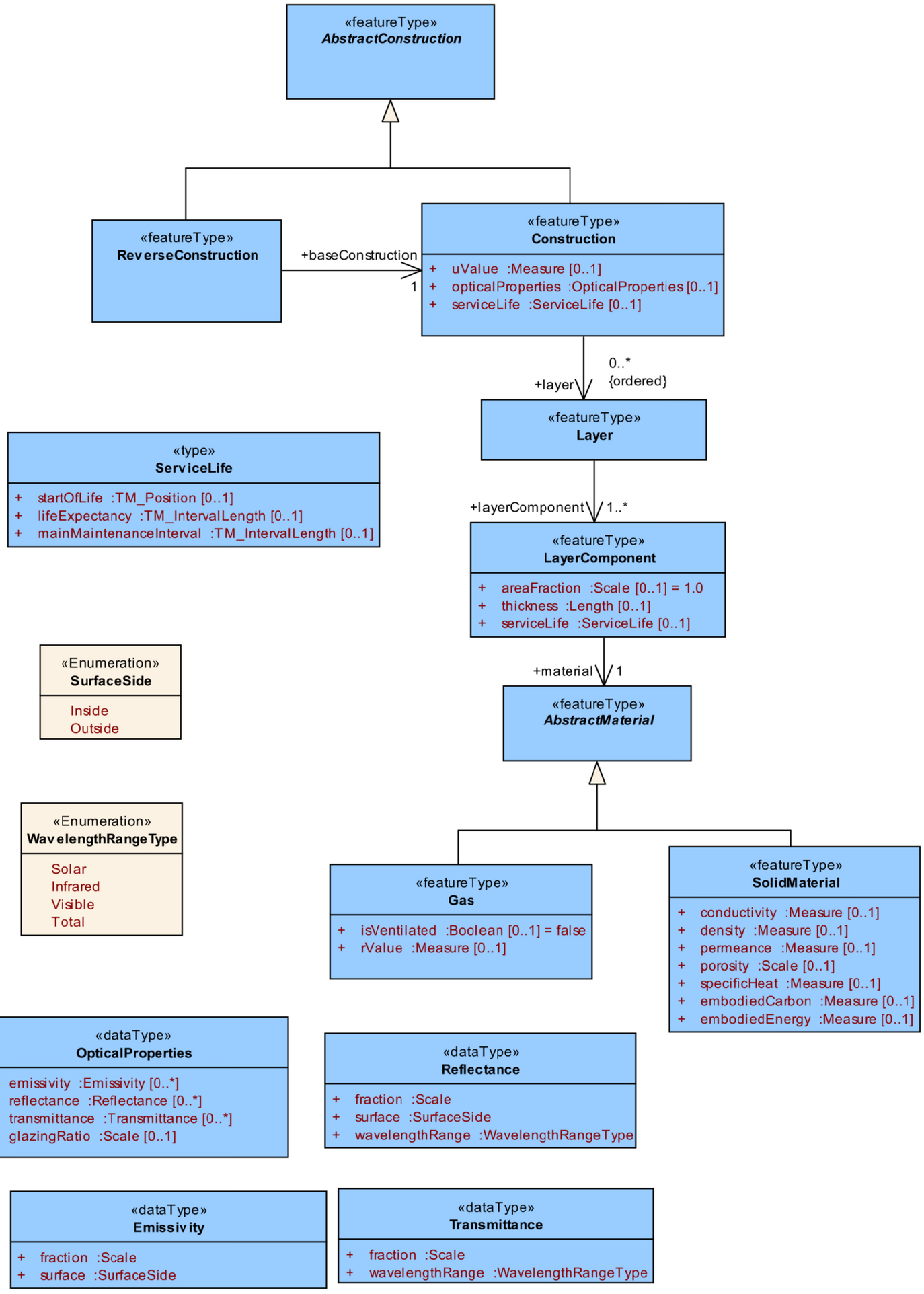


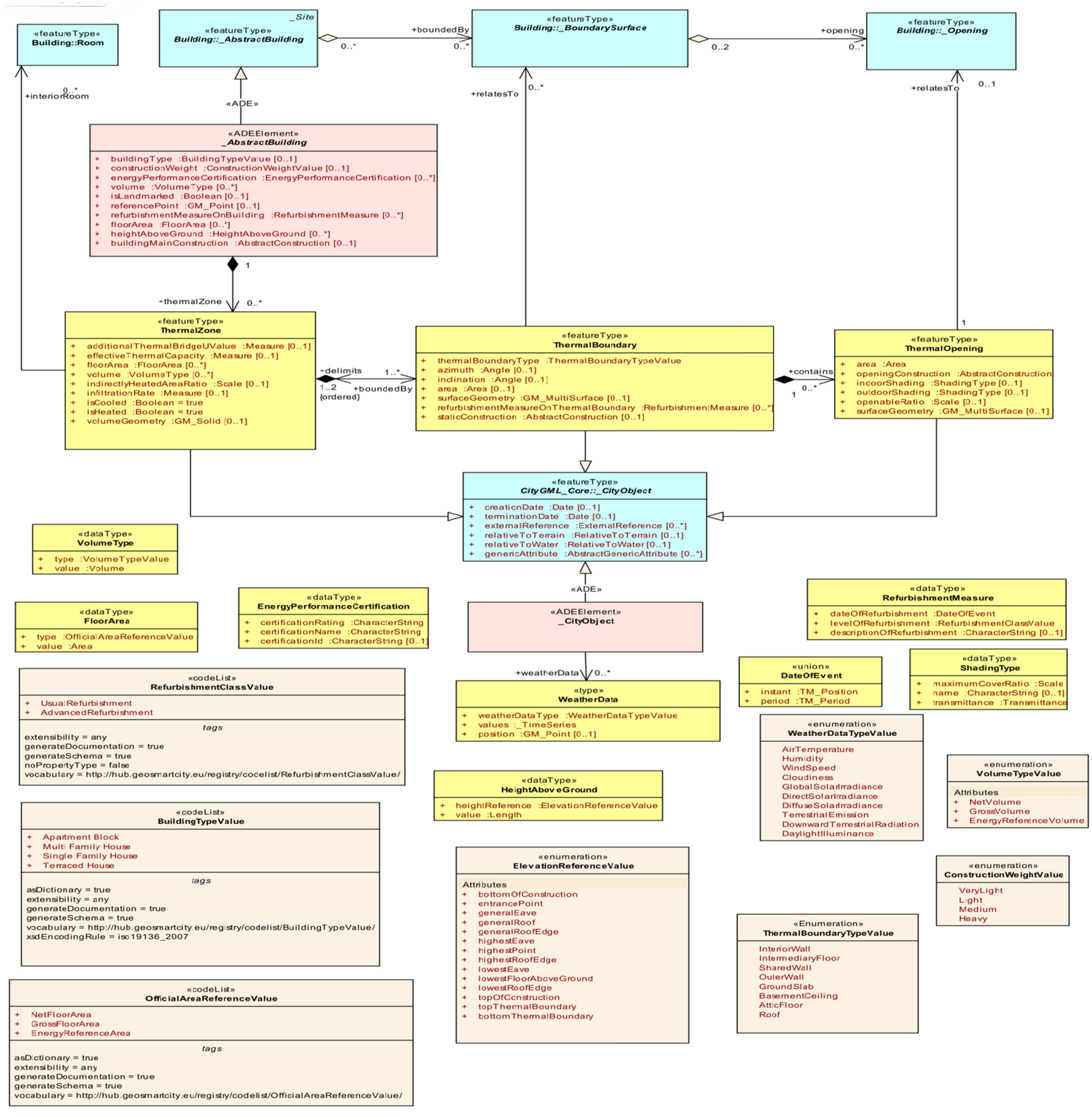
Figure 36: Upon successful installation, the resulting message can be read in the lower part of the window (Output pane)

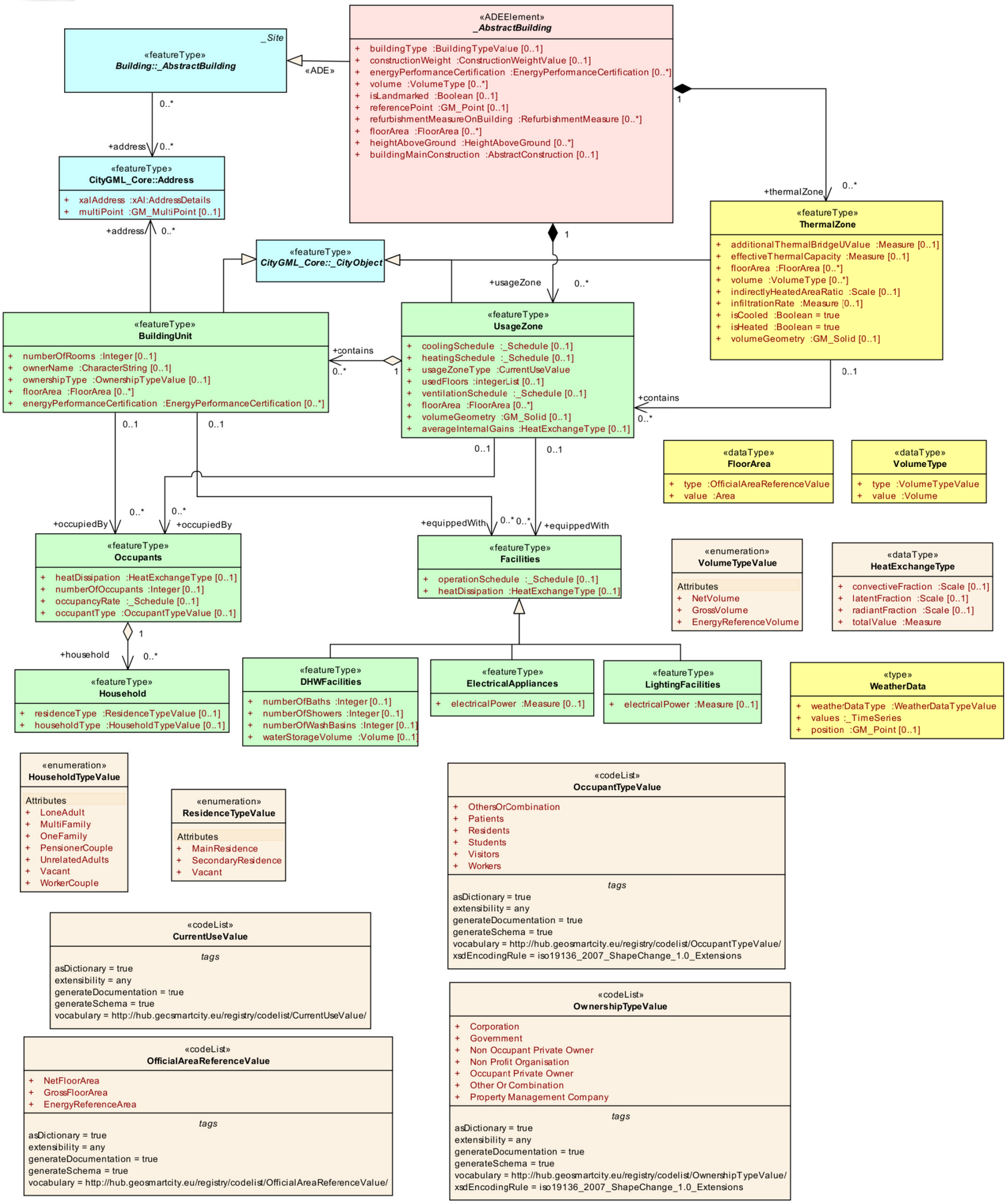
Appendix B: Energy ADE 0.8 UML Diagrams

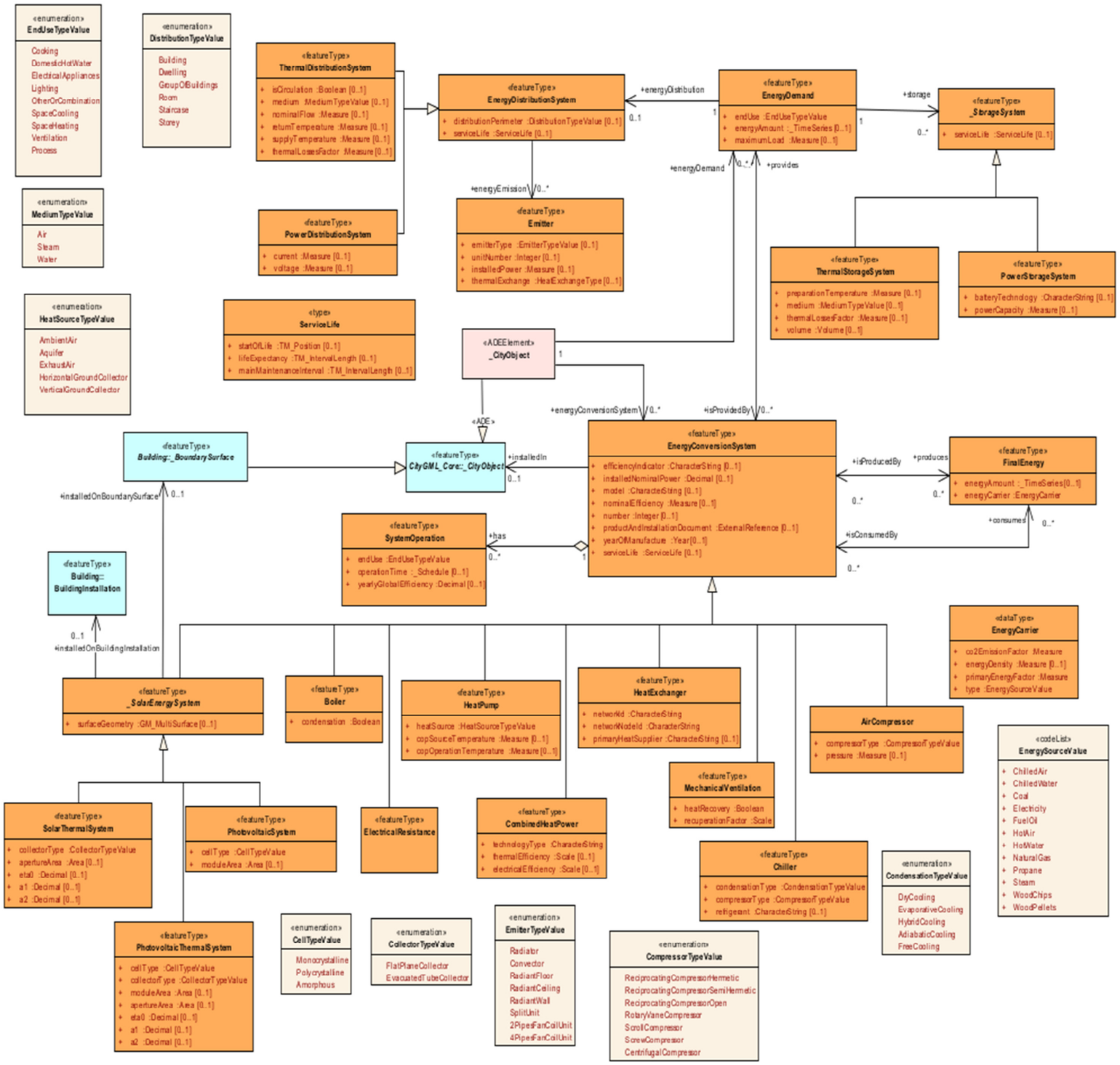
Source: https://github.com/cstb/citygml-energy/blob/v0.8/doc/UML-Diagrams_Energy-ADE.pdf




«featureType»
AbstractConstruction
«featureType»
ReverseConstruction
+baseConstruction
1
«featureType»
Construction
+ uValue :Measure [0..1]
+ opticalProperties :OpticalProperties [0..1]
+ serviceLife :ServiceLife [0..1]
0..*
{ordered}
+layer
«featureType»
Layer
«type»
ServiceLife
+ startOfLife :TM_Position [0..1]
+ lifeExpectancy :TM_IntervalLength [0..1]
+ mainMaintenanceInterval :TM_IntervalLength [0..1]
+layerComponent
1..*
«featureType»
LayerComponent
+ areaFraction :Scale [0..1] = 1.0
+ thickness :Length [0..1]
+ serviceLife :ServiceLife [0..1]
«Enumeration»
SurfaceSide
Inside
Outside
+material
1
«featureType»
AbstractMaterial
«Enumeration»
WavelengthRangeType
Solar
Infrared
Visible
Total
«featureType»
Gas
+ isVentilated :Boolean [0..1] = false
+ rValue :Measure [0..1]
«featureType»
SolidMaterial
+ conductivity :Measure [0..1]
+ density :Measure [0..1]
+ permeance :Measure [0..1]
+ porosity :Scale [0..1]
+ specificHeat :Measure [0..1]
+ embodiedCarbon :Measure [0..1]
+ embodiedEnergy :Measure [0..1]
«dataType»
OpticalProperties
+ emissivity :Emissivity [0..*]
+ reflectance :Reflectance [0..*]
+ transmittance :Transmittance [0..*]
+ glazingRatio :Scale [0..1]
«dataType»
Reflectance
+ fraction :Scale
+ surface :SurfaceSide
+ wavelengthRange :WavelengthRangeType
«dataType»
Emissivity
+ fraction :Scale
+ surface :SurfaceSide
«dataType»
Transmittance
+ fraction :Scale
+ wavelengthRange :WavelengthRangeType


«featureType»
Building::Room
_Site
«featureType»
Building::_AbstractBuilding
+boundedBy
0..*
0..*
«featureType»
Building::_BoundarySurface
0..2
+opening
0..*
«featureType»
Building::_Opening
0..*
+interiorRoom
«ADE»
+relatesTo
0..*
+relatesTo
0..1
«ADEElement»
_AbstractBuilding
+ buildingType :BuildingTypeValue [0..1]
+ constructionWeight :ConstructionWeightValue [0..1]
+ energyPerformanceCertification :EnergyPerformanceCertification [0..*]
+ volume :VolumeType [0..*]
+ isLandmarked :Boolean [0..1]
+ referencePoint :GM_Point [0..1]
+ refurbishmentMeasureOnBuilding :RefurbishmentMeasure [0..*]
+ floorArea :FloorArea [0..*]
+ heightAboveGround :HeightAboveGround [0..*]
+ buildingMainConstruction :AbstractConstruction [0..1]
1
+thermalZone
0..*
«featureType»
ThermalZone
+ additionalThermalBridgeUValue :Measure [0..1]
+ effectiveThermalCapacity :Measure [0..1]
+ floorArea :FloorArea [0..*]
+ volume :VolumeType [0..*]
+ indirectlyHeatedAreaRatio :Scale [0..1]
+ infiltrationRate :Measure [0..1]
+ isCooled :Boolean = true
+ isHeated :Boolean = true
+ volumeGeometry :GM_Solid [0..1]
+delimits
1..*
1..2
+boundedBy
{ordered}
«featureType»
ThermalBoundary
+ thermalBoundaryType :ThermalBoundaryTypeValue
+ azimuth :Angle [0..1]
+ inclination :Angle [0..1]
+ area :Area [0..1]
+ surfaceGeometry :GM_MultiSurface [0..1]
+ refurbishmentMeasureOnThermalBoundary :RefurbishmentMeasure [0..*]
+ staticConstruction :AbstractConstruction [0..1]
+contains
1
0..*
1
«featureType»
ThermalOpening
+ area :Area
+ openingConstruction :AbstractConstruction
+ incoorShading :ShadingType [0..1]
+ outdoorShading :ShadingType [0..1]
+ openableRatio :Scale [0..1]
+ surfaceGeometry :GM_MultiSurface [0..1]
«featureType»
CityGML_Core::_CityObject
+ creationDate :Date [0..1]
+ terminationDate :Date [0..1]
+ externalReference :ExternalReference [0..*]
+ relativeToTerrain :RelativeToTerrain [0..1]
+ relativeToWater :RelativeToWater [0..1]
+ genericAttribute :AbstractGenericAttribute [0..*]
«dataType»
VolumeType
+ type :VolumeTypeValue
+ value :Volume
«ADE»
«ADEElement»
_CityObject
+weatherData
0..*
«dataType»
FloorArea
+ type :OfficialAreaReferenceValue
+ value :Area
«dataType»
EnergyPerformanceCertification
+ certificationRating :CharacterString
+ certificationName :CharacterString
+ certificationId :CharacterString [0..1]
«type»
WeatherData
+ weatherDataType :WeatherDataTypeValue
+ values :_TimeSeries
+ position :GM_Point [0..1]
«dataType»
RefurbishmentMeasure
+ dateOfRefurbishment :DateOfEvent
+ levelOfRefurbishment :RefurbishmentClassValue
+ descriptionOfRefurbishment :CharacterString [0..1]
«union»
DateOfEvent
+ instant :TM_Position
+ period :TM_Period
«dataType»
ShadingType
+ maximumCoverRatio :Scale
+ name :CharacterString [0..1]
+ transmittance :Transmittance
«codeList»
RefurbishmentClassValue
+ UsualRefurbishment
+ AdvancedRefurbishment
tags
extensibility = any
generateDocumentation = true
generateSchema = true
noPropertyType = false
vocabulary = http://hub.geosmartcity.eu/registry/codelist/RefurbishmentClassValue/
«enumeration»
WeatherDataTypeValue
AirTemperature
Humidity
WindSpeed
Cloudiness
GlobalSolarIrradiance
DirectSolarIrradiance
DiffuseSolarIrradiance
TerrestrialEmission
DownwardTerrestrialRadiation
DaylightIlluminance
«enumeration»
VolumeTypeValue
Attributes
+ NetVolume
+ GrossVolume
+ EnergyReferenceVolume
«dataType»
HeightAboveGround
+ heightReference :ElevationReferenceValue
+ value :Length
«codeList»
BuildingTypeValue
+ Apartment Block
+ Multi Family House
+ Single Family House
+ Terraced House
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/BuildingTypeValue/
xsdEncodingRule = iso19136_2007
«enumeration»
ElevationReferenceValue
Attributes
+ bottomOfConstruction
+ entrancePoint
+ generalEave
+ generalRoof
+ generalRoofEdge
+ highestEave
+ highestPoint
+ highestRoofEdge
+ lowestEave
+ lowestFloorAboveGround
+ lowestRoofEdge
+ topOfConstruction
+ topThermalBoundary
+ bottomThermalBoundary
«enumeration»
ConstructionWeightValue
VeryLight
Light
Medium
Heavy
«Enumeration»
ThermalBoundaryTypeValue
InteriorWall
IntermediaryFloor
SharedWall
OuterWall
GroundSlab
BasementCeiling
AtticFloor
Roof
«codeList»
OfficialAreaReferenceValue
+ NetFloorArea
+ GrossFloorArea
+ EnergyReferenceArea
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/OfficialAreaReferenceValue/


«featureType»
Building::_AbstractBuilding
_Site
«ADE»
«ADEElement»
_AbstractBuilding
+ buildingType :BuildingTypeValue [0..1]
+ constructionWeight :ConstructionWeightValue [0..1]
+ energyPerformanceCertification :EnergyPerformanceCertification [0..*]
+ volume :VolumeType [0..*]
+ isLandmarked :Boolean [0..1]
+ referencePoint :GM_Point [0..1]
+ refurbishmentMeasureOnBuilding :RefurbishmentMeasure [0..*]
+ floorArea :FloorArea [0..*]
+ heightAboveGround :HeightAboveGround [0..*]
+ buildingMainConstruction :AbstractConstruction [0..1]
+address
«featureType»
CityGML_Core::Address
+ xalAddress :xAL:AddressDetails
+ multiPoint :GM_MultiPoint [0..1]
+thermalZone
«featureType»
ThermalZone
+ additionalThermalBridgeUValue :Measure [0..1]
+ effectiveThermalCapacity :Measure [0..1]
+ floorArea :FloorArea [0..*]
+ volume :VolumeType [0..*]
+ indirectlyHeatedAreaRatio :Scale [0..1]
+ infiltrationRate :Measure [0..1]
+ isCooled :Boolean = true
+ isHeated :Boolean = true
+ volumeGeometry :GM_Solid [0..1]
«featureType»
CityGML_Core::_CityObject
+usageZone
«featureType»
BuildingUnit
+ numberOfRooms :Integer [0..1]
+ ownerName :CharacterString [0..1]
+ ownershipType :OwnershipTypeValue [0..1]
+ floorArea :FloorArea [0..*]
+ energyPerformanceCertification :EnergyPerformanceCertification [0..*]
+contains
«featureType»
UsageZone
+ coolingSchedule :_Schedule [0..1]
+ heatingSchedule :_Schedule [0..1]
+ usageZoneType :CurrentUseValue
+ usedFloors :integerList [0..1]
+ ventilationSchedule :_Schedule [0..1]
+ floorArea :FloorArea [0..*]
+ volumeGeometry :GM_Solid [0..1]
+ averageInternalGains :HeatExchangeType [0..1]
+contains
«dataType»
FloorArea
+ type :OfficialAreaReferenceValue
+ value :Area
«dataType»
VolumeType
+ type :VolumeTypeValue
+ value :Volume
+occupiedBy
+occupiedBy
«featureType»
Occupants
+ heatDissipation :HeatExchangeType [0..1]
+ numberOfOccupants :Integer [0..1]
+ occupancyRate :_Schedule [0..1]
+ occupantType :OccupantTypeValue [0..1]
+household
«featureType»
Household
+ residenceType :ResidenceTypeValue [0..1]
+ householdType :HouseholdTypeValue [0..1]
+equippedWith
+equippedWith
«featureType»
Facilities
+ operationSchedule :_Schedule [0..1]
+ heatDissipation :HeatExchangeType [0..1]
«enumeration»
VolumeTypeValue
Attributes
+ NetVolume
+ GrossVolume
+ EnergyReferenceVolume
«dataType»
HeatExchangeType
+ convectiveFraction :Scale [0..1]
+ latentFraction :Scale [0..1]
+ radiantFraction :Scale [0..1]
+ totalValue :Measure
«featureType»
DHWFacilities
+ numberOfBaths :Integer [0..1]
+ numberOfShowers :Integer [0..1]
+ numberOfWashBasins :Integer [0..1]
+ waterStorageVolume :Volume [0..1]
«featureType»
ElectricalAppliances
+ electricalPower :Measure [0..1]
«featureType»
LightingFacilities
+ electricalPower :Measure [0..1]
«type»
WeatherData
+ weatherDataType :WeatherDataTypeValue
+ values :_TimeSeries
+ position :GM_Point [0..1]
«enumeration»
HouseholdTypeValue
Attributes
+ LoneAdult
+ MultiFamily
+ OneFamily
+ PensionerCouple
+ UnrelatedAdults
+ Vacant
+ WorkerCouple
«enumeration»
ResidenceTypeValue
Attributes
+ MainResidence
+ SecondaryResidence
+ Vacant
«codeList»
OccupantTypeValue
+ OthersOrCombination
+ Patients
+ Residents
+ Students
+ Visitors
+ Workers
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/OccupantTypeValue/
xsdEncodingRule = iso19136_2007_ShapeChange_1.0_Extensions
«codeList»
CurrentUseValue
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/CurrentUseValue/
«codeList»
OwnershipTypeValue
+ Corporation
+ Government
+ Non Occupant Private Owner
+ Non Profit Organisation
+ Occupant Private Owner
+ Other Or Combination
+ Property Management Company
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/OwnershipTypeValue/
xsdEncodingRule = iso19136_2007_ShapeChange_1.0_Extensions
«codeList»
OfficialAreaReferenceValue
+ NetFloorArea
+ GrossFloorArea
+ EnergyReferenceArea
tags
asDictionary = true
extensibility = any
generateDocumentation = true
generateSchema = true
vocabulary = http://hub.geosmartcity.eu/registry/codelist/OfficialAreaReferenceValue/


«enumeration»
EndUseTypeValue
Cooking
DomesticHotWater
ElectricalAppliances
Lighting
OtherOrCombination
SpaceCooling
SpaceHeating
Ventilation
Process
«enumeration»
DistributionTypeValue
Building
Dwelling
GroupOfBuildings
Room
Staircase
Storey
«featureType»
ThermalDistributionSystem
+ isCirculation :Boolean [0..1]
+ medium :MediumTypeValue [0..1]
+ nominalFlow :Measure [0..1]
+ returnTemperature :Measure [0..1]
+ supplyTemperature :Measure [0..1]
+ thermalLossesFactor :Measure [0..1]
«featureType»
EnergyDistributionSystem
+ distributionPerimeter :DistributionTypeValue [0..1]
+ serviceLife :ServiceLife [0..1]
+energyDistribution
0..1
1
«featureType»
EnergyDemand
+ endUse :EndUseTypeValue
+ energyAmount :_TimeSeries [0..1]
+ maximumLoad :Measure [0..1]
+storage
0..*
«featureType»
_StorageSystem
+ serviceLife :ServiceLife [0..1]
«enumeration»
MediumTypeValue
Air
Steam
Water
«featureType»
PowerDistributionSystem
+ current :Measure [0..1]
+ voltage :Measure [0..1]
+energyEmission
0..*
«featureType»
Emitter
+ emitterType :EmitterTypeValue [0..1]
+ unitNumber :Integer [0..1]
+ installedPower :Measure [0..1]
+ thermalExchange :HeatExchangeType [0..1]
+energyDemand
0..*
+provides
«featureType»
ThermalStorageSystem
+ preparationTemperature :Measure [0..1]
+ medium :MediumTypeValue [0..1]
+ thermalLossesFactor :Measure [0..1]
+ volume :Volume [0..1]
«featureType»
PowerStorageSystem
+ batteryTechnology :CharacterString [0..1]
+ powerCapacity :Measure [0..1]
«enumeration»
HeatSourceTypeValue
AmbientAir
Aquifer
ExhaustAir
HorizontalGroundCollector
VerticalGroundCollector
«type»
ServiceLife
+ startOfLife :TM_Position [0..1]
+ lifeExpectancy :TM_IntervalLength [0..1]
+ mainMaintenanceInterval :TM_IntervalLength [0..1]
«ADEElement»
_CityObject
1
«ADE»
+energyConversionSystem
0..*
+isProvidedBy
0..*
«featureType»
Building::_BoundarySurface
«featureType»
CityGML_Core::_CityObject
+installedIn
0..1
«featureType»
EnergyConversionSystem
+ efficiencyIndicator :CharacterString [0..1]
+ installedNominalPower :Decimal [0..1]
+ model :CharacterString [0..1]
+ nominalEfficiency :Measure [0..1]
+ number :Integer [0..1]
+ productAndInstallationDocument :ExternalReference [0..1]
+ yearOfManufacture :Year [0..1]
+ serviceLife :ServiceLife [0..1]
+isProducedBy
0..*
+produces
0..*
«featureType»
FinalEnergy
+ energyAmount :_TimeSeries [0..1]
+ energyCarrier :EnergyCarrier
+consumes
0..*
+isConsumedBy
0..*
+installedOnBoundarySurface
0..1
«featureType»
SystemOperation
+ endUse :EndUseTypeValue
+ operationTime :_Schedule [0..1]
+ yearlyGlobalEfficiency :Decimal [0..1]
+has
0..*
1
«featureType»
Building::BuildingInstallation
0..1
+installedOnBuildingInstallation
«dataType»
EnergyCarrier
+ co2EmissionFactor :Measure
+ energyDensity :Measure [0..1]
+ primaryEnergyFactor :Measure
+ type :EnergySourceValue
«featureType»
_SolarEnergySystem
+ surfaceGeometry :GM_MultiSurface [0..1]
«featureType»
Boiler
+ condensation :Boolean
«featureType»
HeatPump
+ heatSource :HeatSourceTypeValue
+ copSourceTemperature :Measure [0..1]
+ copOperationTemperature :Measure [0..1]
«featureType»
HeatExchanger
+ networkId :CharacterString
+ networkNodeId :CharacterString
+ primaryHeatSupplier :CharacterString [0..1]
AirCompressor
+ compressorType :CompressorTypeValue
+ pressure :Measure [0..1]
«codeList»
EnergySourceValue
+ ChilledAir
+ ChilledWater
+ Coal
+ Electricity
+ FuelOil
+ HotAir
+ HotWater
+ NaturalGas
+ Propane
+ Steam
+ WoodChips
+ WoodPellets
«featureType»
SolarThermalSystem
+ collectorType :CollectorTypeValue
+ apertureArea :Area [0..1]
+ eta0 :Decimal [0..1]
+ a1 :Decimal [0..1]
+ a2 :Decimal [0..1]
«featureType»
PhotovoltaicSystem
+ cellType :CellTypeValue
+ moduleArea :Area [0..1]
«featureType»
ElectricalResistance
«featureType»
CombinedHeatPower
+ technologyType :CharacterString
+ thermalEfficiency :Scale [0..1]
+ electricalEfficiency :Scale [0..1]
«featureType»
MechanicalVentilation
+ heatRecovery :Boolean
+ recuperationFactor :Scale
«featureType»
Chiller
+ condensationType :CondensationTypeValue
+ compressorType :CompressorTypeValue
+ refrigerant :CharacterString [0..1]
«enumeration»
CondensationTypeValue
DryCooling
EvaporativeCooling
HybridCooling
AdiabaticCooling
FreeCooling
«featureType»
PhotovoltaicThermalSystem
+ cellType :CellTypeValue
+ collectorType :CollectorTypeValue
+ moduleArea :Area [0..1]
+ apertureArea :Area [0..1]
+ eta0 :Decimal [0..1]
+ a1 :Decimal [0..1]
+ a2 :Decimal [0..1]
«enumeration»
CellTypeValue
Monocrystalline
Polycrystalline
Amorphous
«enumeration»
CollectorTypeValue
FlatPlateCollector
EvacuatedTubeCollector
«enumeration»
EmitterTypeValue
Radiator
Convector
RadiantFloor
RadiantCeiling
RadiantWall
SplitUnit
2PipesFanCoilUnit
4PipesFanCoilUnit
«enumeration»
CompressorTypeValue
ReciprocatingCompressorHermetic
ReciprocatingCompressorSemiHermetic
ReciprocatingCompressorOpen
RotaryVaneCompressor
ScrollCompressor
ScrewCompressor
CentrifugalCompressor

Notes

Notes