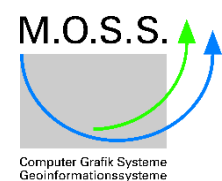
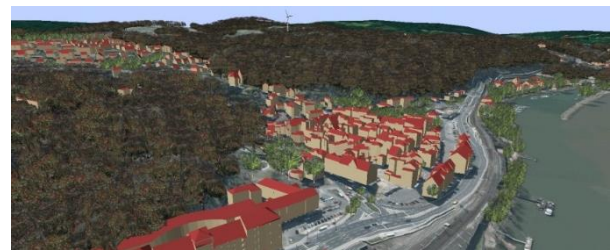
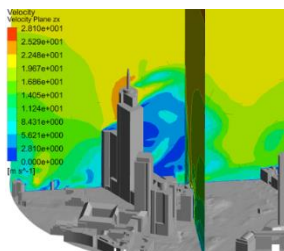
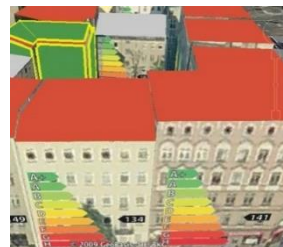
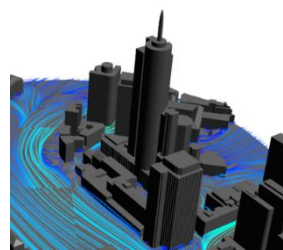
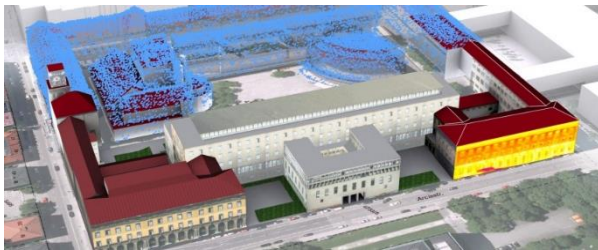


3D City Database for CityGML

Version 3.0.0

Documentation

2015



The images on the cover page were provided by:

- Chair of Photogrammetry and Remote Sensing & Chair of Cartography, Technische Universität München
- Geobasisdaten: © Stadtvermessung Frankfurt am Main
- IDAC Ltd, UK.
- virtualcitySYSTEMS GmbH, Berlin, Germany
- Chair of Geoinformatics, Technische Universität München. Image created based on master thesis work of Matthias Körner, jointly supervised with HTW Dresden
- 3D City Model of Berlin © Berlin partner GmbH
- M.O.S.S. Computer Grafik Systeme GmbH, Taufkirchen, Germany

Active participants in development

Name	Institution	Email
Thomas H. Kolbe Zhihang Yao	Chair of Geoinformatics, Technische Universität München	thomas.kolbe@tum.de zhihang.yao@tum.de
Claus Nagel Felix Kunde	virtualcitySYSTEMS GmbH, Berlin	cnagel@virtualcitysystems.de fkunde@virtualcitysystems.de
Philipp Willkomm György Hudra Arda Müftüoğlu	M.O.S.S. Computer Grafik Systeme GmbH, Taufkirchen, Germany	pwillkomm@moss.de ghudra@moss.de amueftueoglu@moss.de
Javier Herreruella	Institute for Geodesy and Geoinformation Science, Technische Universität Berlin	javier.herreruella@tu-berlin.de

Participants in earlier developments

3D City Database Version 3.0.0 and its tools are based on earlier versions. During the development phase 2006-2012 at the *Institute for Geodesy and Geoinformation Science, TU Berlin*, the following individuals contributed to the development:

Name	Institution	Email
Thomas H. Kolbe Claus Nagel Javier Herreruella Gerhard König Alexandra Lorenz Babak Naderi	Institute for Geodesy and Geoinformation Science, Technische Universität Berlin	gerhard.koenig@tu-berlin.de alexandra.lorenz@tu-berlin.de babak.naderi@telekom.de
Felix Kunde	Universität Potsdam	

During the development phase 2004-2006 at the *Institute for Cartography and Geoinformation, University of Bonn*, the following individuals contributed to the development:

Name	Institution	Email
Thomas H. Kolbe Lutz Plümer Gerhard Gröger Viktor Stroh Jörg Schmittwilken	Institute for Cartography and Geoinformation, University of Bonn	
Andreas Poth Ugo Taddei	lat/ion GmbH, Bonn	

Table of Contents

DISCLAIMER	10
1 INTRODUCTION.....	11
1.1 Main features of 3DCityDB.....	13
1.2 System and design decisions.....	17
1.3 List of changes; new and dropped functionalities.....	18
1.4 Development history.....	19
2 DATA MODELLING AND DATABASE DESIGN	21
2.1 Simplification compared to CityGML 2.0.0	21
2.1.1 Multiplicities, cardinalities and recursions	21
2.1.2 Data type adaptation	22
2.1.3 Project specific classes and class attributes	22
2.1.4 Simplified design of GML geometry classes	22
2.2 UML class diagram.....	22
2.2.1 Geometric-topological Model.....	23
2.2.2 Implicit Geometry.....	24
2.2.3 Appearance Model.....	25
2.2.4 Thematic model	28
2.2.4.1 Core Model.....	28
2.2.4.2 Building model	30
2.2.4.3 Bridge Model.....	34
2.2.4.4 CityFurniture Model.....	37
2.2.4.5 Digital terrain model	38
2.2.4.6 Generic CityObject Model	39
2.2.4.7 LandUse Model	41
2.2.4.8 Transportation Model	41
2.2.4.9 Tunnel Model	43
2.2.4.10 Vegetation Model	47
2.2.4.11 WaterBodies Model.....	48
2.3 Relational database schema.....	50
2.3.1 Mapping rules, schema conventions.....	50
2.3.1.1 Mapping of classes onto tables.....	50
2.3.1.2 Explicit declaration of class affiliation.....	50
2.3.2 Database schema.....	53
2.3.2.1 Core Model.....	53
2.3.2.2 Tables for geometry representation	55
2.3.2.3 Appearance Model	62
2.3.2.4 Building Model.....	66

2.3.2.5	Bridge Model.....	72
2.3.2.6	CityFurniture Model.....	74
2.3.2.7	Digital Terrain Model.....	75
2.3.2.8	Generic City Object Model	77
2.3.2.9	LandUse Model	79
2.3.2.10	Transportation Model	79
2.3.2.11	Tunnel Model	81
2.3.2.12	Vegetation Model	83
2.3.2.13	WaterBody Model	84
2.3.3	Sequences	85
2.3.4	Definition of the CRS for a 3D City Database instance	86
3	IMPLEMENTATION AND INSTALLATION	89
3.1	<i>System requirements.....</i>	89
3.1.1	3D City Database.....	89
3.1.2	Importer/Exporter Tool.....	89
3.2	<i>Installation of the Importer/Exporter and the 3D City Database SQL Scripts</i>	90
3.3	<i>Setting up the database schema.....</i>	92
3.3.1	Oracle.....	93
3.3.2	PostgreSQL.....	96
3.4	<i>Upgrade from previous version of the 3D City Database.....</i>	100
3.4.1	Oracle.....	100
3.4.2	PostgreSQL.....	102
3.4.3	Dropping the old v2.x schema (Oracle and PostgreSQL)	103
4	STORED PROCEDURES AND ADDITIONAL FEATURES	105
4.1	<i>CITYDB_UTIL.....</i>	106
4.2	<i>CITYDB_IDX.....</i>	106
4.3	<i>CITYDB_SRS.....</i>	107
4.4	<i>CITYDB_STAT</i>	107
4.5	<i>CITYDB_DELETE and CITYDB_DELETE_BY_LINEAGE</i>	107
5	IMPORTER / EXPORTER.....	111
5.1	<i>Running and using the Importer / Exporter</i>	111
5.2	<i>Database connections and operations</i>	114
5.2.1	Managing and establishing database connections	114
5.2.2	Executing database operations.....	116
5.3	<i>Importing CityGML files</i>	120
5.4	<i>Exporting to CityGML.....</i>	125
5.5	<i>Exporting to KML/COLLADA.....</i>	127
5.5.1	Support for Point and Line Geometries of GenericCityObject	133

5.5.2	Loading exported models in Google Earth.....	134
5.6	<i>Preferences</i>	137
5.6.1	CityGML import preferences	138
5.6.1.1	Continuation	138
5.6.1.2	gml:id handling.....	139
5.6.1.3	Bounding box	140
5.6.1.4	Address	140
5.6.1.5	Appearance	142
5.6.1.6	Geometry	143
5.6.1.7	Indexes.....	144
5.6.1.8	XML validation	145
5.6.1.9	Import log	147
5.6.1.10	Resources.....	148
5.6.2	CityGML export preferences	151
5.6.2.1	CityGML version	151
5.6.2.2	Bounding box	151
5.6.2.3	CityObjectGroup	153
5.6.2.4	Address	154
5.6.2.5	Appearance	155
5.6.2.6	XLinks	156
5.6.2.7	Resources.....	157
5.6.3	KML/COLLADA export preferences	158
5.6.3.1	General Preferences	158
5.6.3.2	Rendering Preferences.....	161
5.6.3.3	Information Balloon Preferences.....	168
5.6.3.4	Altitude/Terrain Preferences	176
5.6.3.5	General setting recommendations	181
5.6.4	Management of user-defined coordinate reference systems.....	182
5.6.5	General preferences	184
5.6.5.1	Cache	184
5.6.5.2	Import and export path	185
5.6.5.3	Network proxies	185
5.6.5.4	Logging	187
5.6.5.5	Language selection	188
5.7	<i>Map window for bounding box selections</i>	189
5.8	<i>Using the command line interface (CLI)</i>	192
6	IMPORTER / EXPORTER PLUGINS	195
6.1	<i>Introduction to the plugin architecture</i>	195
6.2	<i>Spreadsheet Generator Plugin (SPSHG)</i>	197
6.2.1	Definition.....	197
6.2.2	Plugin installation	197
6.2.3	User Interface	198

6.2.3.1	Main Parameters	198
6.2.3.2	Columns.....	199
6.2.3.3	Content Source	205
6.2.3.4	Output.....	206
6.2.3.5	Export	208
6.2.3.6	Sharing settings	208
6.2.4	Example	210
7	WEB FEATURE SERVICE	217
7.1	<i>System requirements.....</i>	<i>217</i>
7.2	<i>Installation.....</i>	<i>218</i>
7.3	<i>Configuring the Web Feature Service</i>	<i>219</i>
7.3.1	Database settings	220
7.3.2	Capabilities settings.....	223
7.3.3	Feature type settings	224
7.3.4	Operations settings	225
7.3.5	Server settings.....	226
7.3.6	Cache settings	227
7.3.7	Security settings.....	227
7.3.8	Logging settings	228
7.4	<i>Using the Web Feature Service</i>	<i>228</i>
7.4.1	Basic functionality	229
7.4.1.1	WFS operations	229
7.4.1.2	Service URL	230
7.4.1.3	Service bindings	230
7.4.1.4	CityGML feature types.....	231
7.4.1.5	Exception reports.....	231
7.4.2	GetCapabilities operation	231
7.4.3	DescribeFeatureType operation.....	232
7.4.4	ListStoredQueries operation	233
7.4.5	DescribeStoredQuery operation	234
7.4.6	GetFeature operation	235
7.5	<i>Web-based WFS client.....</i>	<i>237</i>
8	REFERENCES.....	239
APPENDIX A	CHANGELOG.....	243
A.1	<i>3D City Database relational schema</i>	<i>243</i>
A.1.1	General changes.....	243
A.1.2	IMPLICIT_GEOMETRY & SURFACE_GEOMETRY.....	243
A.1.3	BUILDING_INSTALLATION & OPENING	244
A.1.4	New tables for new thematic modules BRIDGE & TUNNEL.....	244
A.1.5	CITYOBJECT	244
A.1.6	SOLITARY_VEGETAT_OBJECT	244

A.1.7	RASTER_RELIEF, GRID_COVERAGE & ORTHOPHOTO	244
A.1.8	SURFACE_DATA & TEX_IMAGE & TEXTUREPARAM.....	244
A.2	<i>3D City Database scripts</i>	244
A.3	<i>3D City Database stored procedures</i>	245
A.3.1	General changes.....	245
A.3.2	UTIL package	245
A.3.3	IDX package	246
A.3.4	SRS package	246
A.3.5	STAT package	246
A.3.6	DELETE package	246
A.3.7	DELETE_BY_LINEAGE package	246
A.4	<i>3D City Database Importer/Exporter</i>	246
A.4.1	General changes.....	246
A.4.2	CityGML import.....	247
A.4.3	CityGML export	247
A.4.4	KML/COLLADA export.....	247
A.5	<i>Web Feature Service</i>	247
APPENDIX B	3DCITYDB @ TU MÜNCHEN.....	249
B.1	<i>Interactive Cloud-based 3D Webclient</i>	249
B.2	<i>Research Projects in which 3DCityDB is being used</i>	250
B.3	<i>Current and future work on 3DCityDB</i>	250
APPENDIX C	3DCITYDB @ VIRTUALCITYSYSTEMS	251
C.1	<i>virtualcityDATABASE</i>	251
C.2	<i>virtualcitySUITE – The 3D City Platform</i>	252
APPENDIX D	3DCITYDB @ M.O.S.S.	253
D.1	<i>novaFACTORY at a glance</i>	253
D.2	<i>novaFACTORY 3D GDI</i>	254

Disclaimer

The *3D City Database version 3.0* developed in collaboration of the *Chair of Geoinformatics, Technische Universität München (TUMGI)*, *virtualcitySYSTEMS GmbH*, and *M.O.S.S. Computer Grafik System GmbH* is free software under the GNU Lesser General Public License Version 3.0. See the file LICENSE shipped together with the software for more details. For a copy of the GNU Lesser General Public License see the files COPYING and COPYING.LESSER or visit <http://www.gnu.org/licenses/>.

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

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

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

1 Introduction

Virtual 3D city and landscape models are provided for an increasing number of cities, regions, states, and even countries. They are created and maintained by public authorities like national and state mapping agencies as well as by cadastre institutions and private companies. The 3D topography of urban and rural areas is essential for both visual exploration and a range of different analyses in, for example, the urban planning, environmental, energy, transportation, and facility management sectors.

3D city models are nowadays used as an integrative information backbone representing the relevant urban entities along with their spatial, semantic, and visual properties. They are often created and maintained with full coverage of entire cities, i.e. all real world objects of a specific type like buildings, roads, trees, water bodies, and the terrain are explicitly represented. In most cases the 3D city model objects have well-defined identifiers, which are kept stable during the lifetime of the real world objects and their virtual counterparts. Such complete 3D models are a good basis to organize different types of data and sensors within Smart City projects as they build a stable platform for information linking and enrichment.

In order to establish a common understanding and interpretation of the urban objects and to achieve interoperable access and exchange of complete 3D models including the geometrical, topological, visual, and semantic data, the Open Geospatial Consortium (OGC) has issued the CityGML standard [Kolbe 2009]. CityGML defines a feature catalogue and data model for the most relevant 3D topographic elements like buildings, bridges, tunnels, roads, railways, vegetation, water bodies, etc. The data model is mapped to an XML-based exchange format using OGC's Geography Markup Language (GML).

The 3D City Database (3DCityDB) is a free Open Source package consisting of a database schema and a set of software tools to import, manage, analyse, visualize, and export virtual 3D city models according to the CityGML standard. The database schema results from a mapping of the object oriented data model of CityGML 2.0 to the relational structure of a spatially-enhanced relational database management system (SRDBMS). With this release, 3DCityDB supports the commercial SRDBMS Oracle (with 'Spatial' or 'Locator' license options) and the Open Source SRDBMS PostGIS (which is an extension to the free RDBMS PostgreSQL). 3DCityDB makes use of the specific representation and processing capabilities of the SRDBMS regarding the spatial data elements. It can handle also very large models in multiple levels of details consisting of millions of 3D objects with hundreds of millions of geometries and texture images.

3DCityDB is in use in real life production systems in many places around the world and is also being used in a number of research projects. For example, the cities of Berlin, Potsdam, Munich, Frankfurt, Zurich all keep and manage their virtual 3D city models within an instance of 3DCityDB. The companies virtualcitySYSTEMS and M.O.S.S., who are also partners in development, use 3DCityDB at the core of their commercial products and services to create, maintain, visualize, transform, and export virtual 3D city models. (See Appendix B, Appendix C, and Appendix D) for examples how and where TUM, virtualcitySYSTEMS, and M.O.S.S.

employ 3DCityDB in their projects). Furthermore, the state mapping agencies of all 16 states in Germany store and manage the state-wide collected 3D building models in CityGML LOD1 and LOD2 using 3DCityDB. In 2012 the previous version of 3DCityDB and the developer team received the Oracle Spatial Excellence Award, issued by Oracle USA.

Since 3DCityDB is based on CityGML, interoperable data access from user applications to the database can be achieved in at least two ways:

- 1) by using the included high-performance CityGML Import/Export tool or the included basic Web Feature Service 2.0 in order to exchange the data in CityGML format (Version 2.0 or 1.0), and
- 2) by directly accessing the database tables whose relational structures are fully explained in detail within this document. It is easy to enrich a 3D city model by adding information to the database tables in some user application (using e.g. the database APIs of programming language like C++, Java, Python, or of ETL tools like the Feature Manipulation Engine from Safe Software). The enriched dataset then can be exchanged or archived by exporting the city model to CityGML without information loss. Analogously, 3DCityDB can be used to import a CityGML dataset and then access and work with the city model by directly accessing the database tables from some application programs or ETL software.

The Import/Export tool also provides functionalities for the direct export of 3D visualization models in KML and COLLADA formats. A tiling strategy is supported which allows to visualize even very large 3D city and landscape models in geoinformation systems (GIS) or digital virtual globes like Google Earth or Cesium WebGlobe. The Import/Export tool comes with an API to create further importers, exporters, and database administration tools. One export plugin coming with the software installer package is the so-called ‘Spreadsheet Generator Plugin’ (SPSHG) which allows to export thematic data of 3D objects into tables in CSV and Microsoft Excel format or directly into a Google Spreadsheet within the Cloud.

This document describes the design and the components of the 3D City Database as well as their usage for the new major release 3.0.0 which has been developed and implemented by the three partners in development, namely the *Chair of Geoinformatics at Technische Universität München*, *virtualcitySYSTEMS*, and *M.O.S.S.* The development is continuing the previous work carried out at the *Institute for Geodesy und Geoinformation Science (IGG) of the Berlin University of Technology* and the *Institute for Cartography and Geoinformation (IKG) of the University of Bonn*.

This document has been completely reworked, integrated, extended, and edited from the previous 3DCityDB documentation version 2.0.1 and the latest documentation addendum on 3DCityDB version 2.1.0 and the Importer/Exporter tool version 1.6.0. Some figures and texts are cited from the OpenGIS City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0 [Gröger et al. 2012].

1.1 Main features of 3DCityDB

Many (but not all) of the features referring to object modelling and representation are implied by following the CityGML standard 2.0.0 issued by the Open Geospatial Consortium.

- **CityGML 2.0.0 and 1.0.0 compliant database:** The implementation defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. Included are generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties. This thematic information goes beyond graphic exchange formats and allows to employ virtual 3D city models for sophisticated analysis tasks in different application domains.
- **Implementation on the basis of a spatially-enhanced relational database management system (Oracle 10G R2 or higher with Spatial/Locator option; PostgreSQL 9.1 or higher with PostGIS extension 2.0 or higher):** For the representation of all vector and grid geometry the built-in data types provided by the SRDBMS are used exclusively. This way, special solutions are avoided and different geoinformation systems, CAD/BIM systems, and ETL software systems can directly access (read and write) the geometry objects stored in the SRDBMS.
- **Tool for importing and exporting CityGML data:** The included Importer/Exporter software tool allows for high performance importing and exporting of CityGML datasets according to CityGML versions 2.0.0 and 1.0.0. The tool allows processing of very large datasets (>> 4 GB), even if they include XLinks between CityGML features or XLinks to 3D GML geometry objects. The multi-threaded programming exploits multiprocessor systems or multikernel CPUs to speed up the processing of complex XML-structures, resulting in high performance database access. Objects can be filtered during import or export according to spatial regions (bounding box) and their object IDs, feature types, and names. Bounding boxes can be interactively selected using a map window based on OpenStreetMap (OSM). A tiling strategy is implemented in order to support the export of very large datasets. In case of a high number of texture images they can be automatically distributed in a configurable number of subdirectories in order to avoid large directories with millions of files which can render a Windows operating system unresponsive. The Importer can also validate CityGML files and can be configured to only import valid features. The Importer/ Exporter tool can be run in interactive or batch mode.
- **Tool for exporting visualization models in KML and COLLADA formats:** This tool exports city models from the 3D city database in KML and COLLADA formats which can directly be viewed and interactively explored in geoinformation systems (GIS) or digital virtual globes like Google Earth or Cesium WebGL Virtual Globe. A tiling strategy is supported where only tiles in the vicinity of the viewer's location are being loaded facilitating the visualization of even very large 3D city and landscape models. Information balloons for all objects can be configured by the user.

- Tool for exporting data to spreadsheets:** The ‘Spreadsheet Generator’ (SPSHG) allows exporting thematic data of 3D objects into tables in CSV and Microsoft Excel format or directly into a Google Spreadsheet within the Google Document Cloud. For every selected geobject one row is being exported where the first column always contains the GML ID value of the respective object. The further columns can be selected by the user. This tool can be used to export attribute data from e.g. buildings like the class, function, usage, roof type, address, and further generic attributes that may contain information like the building energy demand, potential solar energy gain, noise level on the facades etc. The spreadsheet rows can be linked to the visualization model generated by the KML/COLLADA Exporter. This is shown in Appendix B.
- Web Feature Service (WFS) 2.0:** 3DCityDB comes with an OGC compliant implementation of a basic WFS 2.0 allowing web-based access to the 3D city objects stored in the database. WFS clients can directly connect to this interface and retrieve 3D content for a wide variety of purposes. The implementation currently satisfies the *Simple WFS* conformance class. An implementation of a full, transactional WFS is commercially available from one of the development partners, see Appendix C.
- Support of different kinds of multi-representations: Levels of detail, different appearances, (and with Oracle RDBMS only) planning versions and history:** Every geobject as well as the DTM can be represented in five different resolution or fidelity steps (Levels of Detail, LOD). With increasing LOD, objects do not only obtain a more precise and finer geometry, but do also gain a thematic refinement. Different appearance data may be stored for each city object. Appearance relates to any surface-based theme, e.g. infrared radiation or noise pollution, not just visual properties. Consequently, data provided by appearances can be used as input for both presentation and analysis of virtual 3D city models. The database supports feature appearances for an arbitrary number of themes per city model. Each LOD of a feature can have individual appearances. Appearances can represent – among others – textures and georeferenced textures. All texture images can be stored in the database. The version and history management employs Oracle’s Workspace Manager and, hence, is only available for 3DCityDB instances running on an Oracle RDBMS. It is largely transparent to application programs that work with the database. Procedures saved within the database (Stored Procedures) are provided, which allow for the management of planning alternatives and versions via application programs.
- Complex digital terrain models:** DTMs may be represented in four different ways in CityGML and therefore also in the 3D city database: regular grids, triangular irregular networks (TINs), 3D mass points and 3D break lines. For every level of detail a complex DTM consisting of any number of DTM components and DTM types can be defined. Besides, it is possible to combine certain kinds of DTM representations for the same geographic area with each other (e.g. mass points and break lines or grids and break lines). In Oracle Spatial (but not Locator) Grid-based DTMs may be of arbitrary size and are composed from separate tiles to a single overall grid using the Oracle GeoRaster

functionality. Please note that the Import/Export tool provides functions to read and write TIN, mass point, and break line DTM components, but not for raster based DTMs. GeoRaster data would have to be imported and exported using other tools from e.g. Oracle, ESRI, or Safe Software.

- **Complex city object modelling:** The representation of city objects in the 3D city database ranges from coarse models to geometrically and semantically fine grained structures. The underlying data model is a complete realization of the CityGML data model for the levels of detail (LOD) 0 to 4. For example, buildings can be represented by simple, monolithic objects or can consist of an aggregation of building parts. Extensions of buildings, like balconies and stairs, can be classified thematically and provided with attributes just as single surfaces can be. LOD4 completes a LOD3 model by adding interior structures for 3D objects. For example, LOD4 buildings are composed of rooms, interior doors, stairs, and furniture. This allows among other things to select the floor space of a building, so that it can later be used e.g. to derive SmartBuildings or to form 3D solids by extrusion [Döllner et al. 2005]. Buildings can be assigned addresses that are also stored in the 3D city database. Their implementation refers to the OASIS xAL Standard, which maps the address formats of the different countries into a unified XML schema. In order to model whole complexes of buildings, single buildings can be aggregated to form special building groups. The same complex modelling applies to the other CityGML feature types like bridges, tunnels, transportation and vegetation objects, and water bodies.
- **Representation of generic and prototypical 3D objects:** Generic objects enable the storage and management of 3D geoobjects that are not explicitly modelled in CityGML yet, for example dams or city walls, or that are available in a proprietary file format only. This way, files from other software systems like architecture or computer graphics programs can be imported directly into the database (without interpretation). However, application systems that would like to use these data must be able to interpret the corresponding file formats after retrieving them back from the 3D geodatabase. Prototypical objects are used for memory-efficient management of objects that occur frequently in the city model and that do not differ with respect to geometry and appearance. Examples are elements of street furniture like lanterns, road signs or benches as well as vegetation objects like shrubs, certain tree types etc. Every instance of a prototypical object is represented by a reference to the prototype, a base point and a transformation matrix for scaling, rotating and translating the prototype. The geometries (and appearances like textures, colors etc.) of generic objects as well as prototypes can be stored either using the geometry datatype of the spatial database management system (Oracle Spatial/Locator or PostGIS) or in proprietary file formats. In the latter case a single file may be saved for every object, but the file type (MIME type), the coordinate transformation matrix that is needed to integrate the object into the world coordinate reference system (CRS) as well as the target CRS have to be specified.

- **Extendable object attribution:** All objects in the 3D geodatabase can be augmented with an arbitrary number of additional generic attributes. This way, it is possible to add further thematic information as well as further spatial properties to the objects at any time. In combination with the concept of generic 3D objects this provides a highly flexible storage option for object types which are not explicitly defined in the CityGML standard. Every generic attribute consists of a triple of attribute name, data type, and value. Supported data types are: string; integer and floating-point numbers; date; time; binary object (BLOB, e.g. for storing a file); geometry object according to the specific geometry data type of Oracle or PostGIS respectively; simple, composite, or aggregate 3D solids or surfaces. Please note that generic attributes of type BLOB or geometry are not allowed as generic attributes in CityGML (and will, thus, not be exported by the CityGML exporter). However, it may be useful to store binary data associated with the individual city objects, for example, to store derived 3D computer graphics representations.
- **Free, also recursive grouping of geoobjects:** Geoobjects can be grouped arbitrarily. The aggregates can be named and may also be provided with an arbitrary number of generic attributes (see above). Object groups may also contain object groups, which leads to nested aggregations of arbitrary depth. In addition, for every object of an aggregation, its role in the group can be specified explicitly (qualified association).
- **External references for all geoobjects:** All geoobjects can be provided with an arbitrary number of references to corresponding objects in external data sources (i.e. hyperlinks / linked data). For example, in case of building objects this allows to store e.g. the IDs of the corresponding objects in official cadasters, digital landscape models, or Building Information Models (BIM). Each reference consists of an URI to the external data store or database and the corresponding object ID or URI within that external data store or database.
- **Flexible 3D geometries:** The geometry of most 3D objects can be represented through the combination of solids and surfaces as well as any - also recursive - aggregation of these elements. Each surface may have attached different textures and colors on both its front and back face. It may also comprise information on transparency. Additional geometry types (any geometry type supported by the spatial database management system Oracle Spatial/Locator or PostGIS) can be added to the geoobjects by using generic attributes.
- **Open Source and Platform Independence:** The entire software is freely accessible to the interested public. The release was carried out under the LGPL Version 3, which allows including 3DCityDB in commercial systems. See the GNU Lesser General Public License at <http://www.gnu.org/licenses/lgpl-3.0.html> for more details. Both the Importer/Exporter tool and the Web Feature Service are implemented in Java and can be run on different platforms and operating systems.

1.2 System and design decisions

3D City Database is implemented as a relational database schema using the spatial datatypes provided by a spatially-enhanced relational database management system (SRDBMS). Above, external software applications and database stored procedures are provided working on this database schema. Since only 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) offer comprehensive support for 3D spatial data, the 3D City Database schema is being provided for these two systems only.

In addition to the general advantages arising from the usage of a widely used relational database management system (RDBMS), both Oracle Spatial/Locator and PostgreSQL/ PostGIS offer some important performance characteristics that allow an efficient implementation of the required functionalities:

- Both RDBMS support spatial data types with coordinates ranging from 2D to 4D. Spatial indexes and filters can be 2D or 3D allowing for efficient spatial selections in very large city models. Furthermore, the spatial data types are supported by a number of commercial and Open Source GIS that provide a database connection as for example ESRI's ArcGIS/ArcSDE or Safe Software's Feature Manipulation Engine (FME). This enables such systems to directly access the data stored in the 3D geodatabase.
- Rules can be implemented using stored procedures and trigger mechanisms which propagate updates of objects to likewise affected objects in the database (transparent for the user).

The data model of the 3D City Database is based on the CityGML 2.0.0 standard. The object-oriented data model of CityGML has been mapped to a purely relational data model with the exception that geometry objects are mapped to the spatial datatypes provided by the SDBMS. In order to achieve high performance for data manipulations and queries the mapping was done manually with a number of optimizations. A few simplifying assumptions were made regarding the usage of the CityGML concepts in the real world helping to increase performance. These are documented in chapter 2.

Surface-based geometries like Polygons, TINs, MultiSurfaces as well as Solids are stored in a special way: they are decomposed into their primitive surfaces and each surface is stored as an individual tuple in one big surface table. The reason for this is that each surface can be assigned multiple appearances (e.g. textures) in CityGML and, thus, each appearance must be explicitly linkable to the corresponding surface. For Solids also the solid geometry objects are stored in addition to their decomposed boundary surfaces allowing to apply spatial operations on them like the computation of the volume.

The provided software tools like the Importer/Exporter application are implemented in the Java language in order to be platform independent. The tools have been confirmed to run under Microsoft Windows, Linux, and Apple Mac OS X. High performance is achieved by exploiting multi-threading on multiprocessor or multi-core CPU systems.

1.3 List of changes; new and dropped functionalities

Significant changes with respect to the previous software versions (3DCityDB 2.1.0, Importer/Exporter 1.6.0):

- Updated and extended database schema supporting CityGML 2.0. CityGML 1.0 datasets can be converted on-the-fly into CityGML 2.0. Export and import of CityGML 1.0 datasets are still supported.
- Adaption and extension of the CityGML Importer/Exporter as well as of the KML/COLLADA Exporter to the new database schema / CityGML 2.0
- Performance improvements when reading and writing large CityGML datasets over a network.
- Improved support of 3D city models which use texture atlases regarding a more compact storage and a higher import, export, and visualization performance.
- Improved handling of 3D models with large coordinate values in the KML/COLLADA Exporter.
- Both Oracle and PostGIS databases are supported by the same Import/Export tool (there are no longer separate versions of the Import/Export tool).
- Unified concept for the storage of image data in Oracle and PostGIS.
- Completely reworked, extended, and updated documentation.

New features and functionalities:

- Support of all new CityGML 2.0 feature types like bridges and tunnels with respect to the previous version of CityGML. Support of CityGML 2.0's grouping concept for generic attributes.
- CityGML Solid geometry objects are now stored as 3D solid geometries in Oracle (starting from version 11g) and PostGIS in addition to their decomposition into the individual boundary surfaces.
- This software package comes with an implementation of an OGC compliant basic Web Feature Service 2.0.
- The newly included 'Spreadsheet Generator' plugin allows exporting data from a 3D City Database instance into tables. Supported output formats are Microsoft Excel, CSV, and Google Spreadsheets.
- Database scripts for Oracle and PostGIS for the migration of a 3DCityDB database version 2.1.0 to the new version 3.0.0.
- Support of Oracle Spatial and Locator licensing options.
- Schema files of the database schemas are included for Oracle and PostGIS
- A special database procedure allows performing a change of the spatial reference system of a 3D city database. Coordinate transformations will be carried out in-situ.
- Support of GenericCityObjects having point and/or line geometries in the KML/COLLADA Exporter.

Dropped items or functionalities:

- The ‘Merging’ plugin of the previous Importer/Exporter version 1.6.0 was not updated to the new database schema and has been removed.
- The database table ORTHOPHOTO which was part of the previous version of the 3DCityDB schema was dropped. It was not supported by the previous versions of the Import/Export tool anyway.
- The tool for importing and exporting raster data for Oracle Spatial is no longer supported. Please use third party software instead.
- The Oracle and PostgreSQL scripts for creating a read-only user have been removed from this release. It is planned to include a reworked version of these scripts in the next minor release.

1.4 Development history

The development of the 3D City Database was always closely related to the development of the CityGML standard [Kolbe & Gröger 2003]. It was started back in 2003 by *Dr. Kolbe* and *Prof. Plümer* at the *Institute for Cartography and Geoinformation at University of Bonn*. In the period from November 2003 to December 2005 the official virtual 3D city model of Berlin, commissioned by *The Berlin Senate* and *Berlin Partner GmbH*, was developed within a pilot project funded by the European Union [Plümer et al. 2005, Berlin 3D]. Since then, the model has been playing a central role in the three-dimensional spatial data infrastructure of Berlin and opened up a multitude of applications for the public and private sector alike. As an example the virtual city model is successfully used for presentation of the business location, its urban development combined with application related information to politicians, investors, and the public in order to support civic participation, provide access to decision-making content, assist in policy-formulation, and control implementation processes [Döllner et al. 2006]. 3DCityDB was key in demonstrating the real world usage of CityGML to the Open Geospatial Consortium on the one hand, and the practical usability and versatility of CityGML to the city of Berlin on the other hand. This first development phase was carried out by University of Bonn in collaboration with the company *lat/lon GmbH*. Oracle Spatial was the only supported SDBMS in that phase and the next (3DCityDB Versions 0.2 up to 1.3).

Within the framework *Europäische Fonds für regionale Entwicklung* (EFRE II) the project *Geodatenmanagement in der Berliner Verwaltung – Amtliches 3D-Stadtmodell für Berlin* allowed for upgrading the official 3D city model based on the former CityGML specification draft 0.4.0 in the year 2007. The developments were carried out by the *Institute for Geodesy and Geoinformation Science (IGG) of the Berlin University of Technology* (where Kolbe became full professor for Geoinformation Science in 2006) on behalf of the *Berliner Senatsverwaltung für Wirtschaft, Arbeit und Frauen* and the *Berlin Partner GmbH* (former Wirtschaftsförderung Berlin International). The relational database model (3DCityDB versions 1.4 up to 1.8) was implemented and evaluated in cooperation with *3DGeo GmbH* (later bought by *Autodesk GmbH*) in Potsdam. A special database interface for LandXPlorer was provided by 3DGeo / Autodesk. Later on, a first version of the Java based CityGML Importer/Exporter was developed [Stadler et al. 2009].

In August 2008, CityGML 1.0.0 became an adopted standard of the Open Geospatial Consortium (OGC). In the follow-up project *Digitaler Gestaltplan Potsdam* starting in 2010 the 3DCityDB version 2 was developed which brought a full support for all CityGML 1.0.0 feature types. The KML/COLLADA exporter was added as well as the ‘Matching’ plugin. This project was carried out by *IGG of TU Berlin* on behalf of and in collaboration with the company *virtualcitySYSTEMS (VCS)* in Berlin. In 2012 the developer team at *TU Berlin* received the *Oracle Spatial Excellence Award for Education and Research* from *Oracle USA* for our work on 3DCityDB. Also in 2012 3DCityDB was ported to PostgreSQL/PostGIS by *Felix Kunde*, a master student from the *University of Potsdam*, who did his master thesis in collaboration with *IGG* [Kunde 2013].

In August 2012, CityGML 2.0.0 became an adopted standard of the Open Geospatial Consortium (OGC). In September 2012, Prof. Kolbe moved from IGG, TU Berlin to the *Chair of Geoinformatics at Technische Universität München (TUM)*. The companies *virtualcitySYSTEMS GmbH* in Berlin and *M.O.S.S. GmbH* in Taufkirchen (near Munich) have also been using the 3D City Database in their commercial projects for a number of years. In this context, the Chair of Geoinformatics at TUM and the companies *virtualcitySYSTEMS* and *M.O.S.S.* signed an official collaboration agreement on the joint further development of 3DCityDB and its tools. The work on the new major release version 3.0.0 began in 2013 when Dr. Nagel finished his PhD and went to the company VCS.

2 Data Modelling and Database Design

In this section the slightly simplified data model with respect to CityGML is described at the conceptual level using UML class diagrams. These diagrams form the basis for the implementation-dependent realization of the model with a relational database system which is presented in section 2.3. However, UML diagrams may also form the basis for other implementations e.g. for the definition of an exchange format based on XML or GML. The UML diagrams of the 3D city model are depicted in section 2.2.

2.1 *Simplification compared to CityGML 2.0.0*

CityGML is a common information model for 3D urban objects and provides a comprehensive and extensible representation of the objects. It is explained in detail in the CityGML specification [Gröger et al. 2008, Gröger et al. 2012] and [Kolbe 2009]. An analysis of the previous versions of the 3D City Database indicated that for the data collected and processed a less complex schema is sufficient. Using a simplified schema usually allows improving system performance. Therefore the first task was related to database design aspects with respect to adjusting the comprehensive CityGML features. As result a simplified database schema was generated, allowing an optimized workflow and guaranteeing efficient processing time. The related UML-diagrams were discussed and coordinated with the project partners and translated into the relational schema. Based on this work the SQL scripts for setting up the Oracle and PostgreSQL database schema were generated. Please note, that all test CityGML datasets (versions 1.0.0 and 2.0.0) from the CityGML homepage (and others) can be stored and managed without restrictions with this simplified database schema.

2.1.1 Multiplicities, cardinalities and recursions

Simplifications with respect to the CityGML specification were made as follows:

- **Multiplicities of attributes**

Attributes with a variable amount of occurrences (*) are substituted by a data type enabling the storage of arbitrary values (e.g. data type String with a predefined separator) or by an array with a predefined amount of elements representing the number of objects that participate in the association. This means that object attributes can be stored in a single column.

- **Cardinalities and types of relationships**

n:m relations require an additional table in the database. This table consists of the primary keys of both elements' tables which form a composite primary key. If the relation can be restricted to a 1:n or n:1 relationship the additional table can be avoided. Therefore all n:m relations in CityGML were checked for a more restrictive definition. This results in simplified cardinalities and relations.

- **Simplified treatment of recursions**

Some recursive relations are used in the CityGML data model. Recursive database queries may cause high cost, especially if the amount of recursive steps is unknown. In order to guarantee good performance, implementation of recursive associations receive two additional columns which contain the ID of the parent and of the root element. For

example, if all building parts related to a specific building are queried, only those tuples containing the ID of the building as root element have to be selected. Thus, typical queries concerning object geometry remain high-performance.

2.1.2 Data type adaptation

Data types specified in CityGML were substituted by data types which allow an effective representation in the database. Strings for example are used to represent code types and number vectors; GML geometry types were changed to the database geometry data type. Matrices are stored each one as String data type, with values listed in a row-major sequence separated by spaces.

2.1.3 Project specific classes and class attributes

The 3D city database may contain some classes for representation of project specific metadata, version control and attributes for representation of additional project specific information. Since this information is represented in the CityGML specification differently or even not at all, appropriate classes and class attributes are added or respectively adopted.

2.1.4 Simplified design of GML geometry classes

Spatial properties of features are represented by objects of GML3's geometry model based on the ISO 19107 standard 'Spatial Schema' [Herring 2001], representing 3D geometry according to the well-known Boundary Representation (B-Rep, cf. [Foley et al. 1995]). Actually only a subset of the GML3 geometry package is used. Moreover, for 2D and 3D surface-based geometry types a simpler but equally powerful model is used: These geometries are stored as polygons, which are aggregated to MultiSurfaces, CompositeSurfaces, TriangulatedSurfaces, Solids, MultiSolids, as well as CompositeSolids.

2.2 UML class diagram

The following pages cite several parts of the CityGML specification [Gröger et al., 2012] which are necessary for a better understanding. Main focus is put on explaining the customization and the differences to the CityGML standard.

Design decisions in the model are explicitly visualised within the UML diagrams. Following models are presented in detail:

- Geometric-topological model
- Appearance model
- Thematic Model
 - CityGML Core
 - Building model
 - Bridge model
 - City furniture
 - Digital Terrain Model
 - Generic objects and attributes
 - Land use
 - Transportation objects

- Tunnel model
- Water bodies
- Vegetation objects

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. N to m relations, which only can be represented by additional tables, are represented as green blocks.

2.2.1 Geometric-topological Model

The geometry model of CityGML consists of primitives, which may be combined to form complexes, composite geometries or aggregates. A zero-dimensional object is modelled as a *Point*, a one-dimensional as a *_Curve*. A curve is restricted to be a straight line, thus only the GML3 class *LineString* is used.

Combined geometries can be aggregates, complexes or composites of primitives (see illustration in figure 1). In an *Aggregate*, the spatial relationship between components is not restricted. They may be disjoint, overlapping, touching, or disconnected. GML3 provides a special aggregate for each dimension, a *MultiPoint*, a *MultiCurve*, a *MultiSurface* or a *MultiSolid*. In contrast to aggregates, a *Complex* is topologically structured: its parts must be disjoint, must not overlap and are allowed to touch, at most, at their boundaries or share parts of their boundaries. A *Composite* is a special complex provided by GML3. It can only contain elements of the same dimension. Its elements must be disjoint as well, but they must be topologically connected along their boundaries. A *Composite* can be a *CompositeSolid*, a *CompositeSurface*, or *CompositeCurve*.

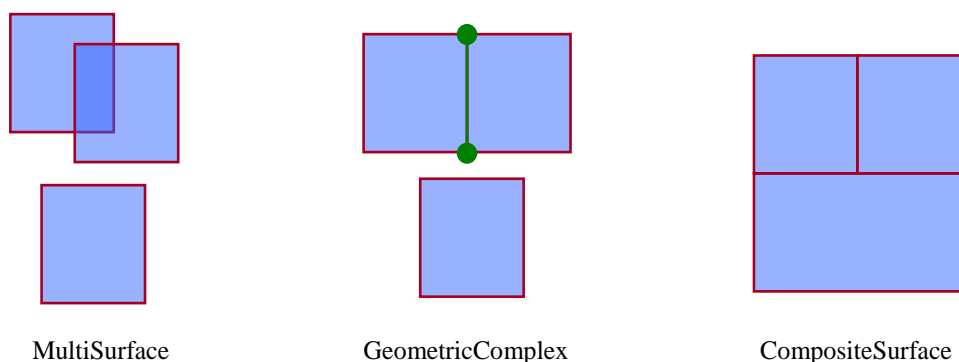


Figure 1: Different types of aggregated geometries [Gröger et al., 2012]

The modelling of two-dimensional and three-dimensional geometry types is handled in a simplified way. All surface-based geometries are stored as polygons, which are aggregated to *MultiSurfaces*, *CompositeSurfaces*, *TriangulatedSurfaces*, *Solids*, *MultiSolids*, as well as *CompositeSolids* accordingly. This simplification substitutes the more complex representation used for those GML geometry classes in grey blocks in Figure 2. Mapping the UML diagram to the relational schema now requires only one table (SURFACE_GEOMETRY), which is explained in chapter 2.3.2.2.

feature occurs in the 3D city model. Each occurrence is represented by a link to the prototypic shape geometry (in a local Cartesian coordinate system), by a transformation matrix that is multiplied with each 3D coordinate of the prototype, and by an anchor point denoting the base point of the object in the world coordinate reference system. The concept of implicit geometries is similar to the well-known concept of *primitive instancing* used for the representation of *scene graphs* in the field of computer graphics [Foley et al. 1995].

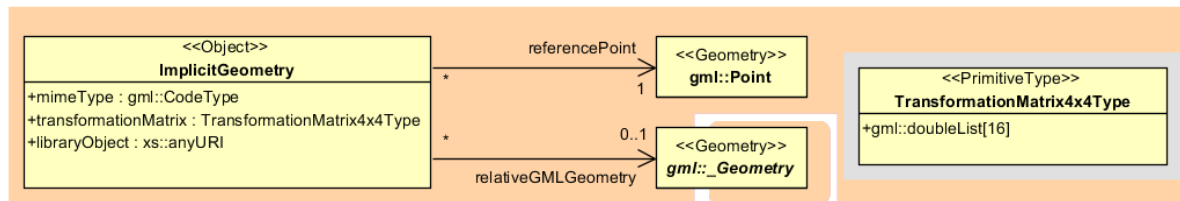


Figure 3: Implicit Geometry model

Implicit geometries may be applied to features from different thematic fields in order to geometrically represent the features within a specific level of detail (LOD). Thus, each CityGML thematic extension module (like Building, Bridge, and Tunnel etc.) may define spatial properties providing implicit geometries for its thematic classes.

The shape of an implicit geometry can be represented in an external file with a proprietary format, e.g. a VRML file, a DXF file, or a 3D Studio MAX file. The reference to the implicit geometry can be specified by an URI pointing to a local or remote file, or even to an appropriate web service. Alternatively, a GML3 geometry object can define the shape. This has the advantage that it can be stored or exchanged inline within the CityGML dataset. Typically, the shape of the geometry is defined in a local coordinate system where the origin lies within or near to the object's extent. If the shape is referenced by an URI, also the MIME type of the denoted object has to be specified (e.g. "model/vrml" for VRML models or "model/x3d+xml" for X3D models).

The implicit representation of 3D object geometry has some advantages compared to the explicit modelling, which represents the objects using absolute world coordinates. It is more space-efficient, and thus more extensive scenes can be stored or handled by a system. The visualization is accelerated since 3D graphics hardware supports the scene graph concept. Furthermore, the usage of different shape versions of objects is facilitated, e.g. different seasons, since only the library objects have to be exchanged.

2.2.3 Appearance Model

Information about a surface's appearance, i.e. observable properties of the surface, is considered an integral part of virtual 3D city models in addition to semantics and geometry. Appearance relates to any surface-based theme, e.g. infrared radiation or noise pollution, not just visual properties and can be represented by – among others – textures and georeferenced textures. Appearances are supported for an arbitrary number of themes per city model. Each LoD of a feature can have individual appearances. Each city object or city model respectively may store its own appearance data. Therefore, the base CityGML classes *_CityObject* and *CityModel* contain a relation *appearance* and *appearanceMember* respectively.

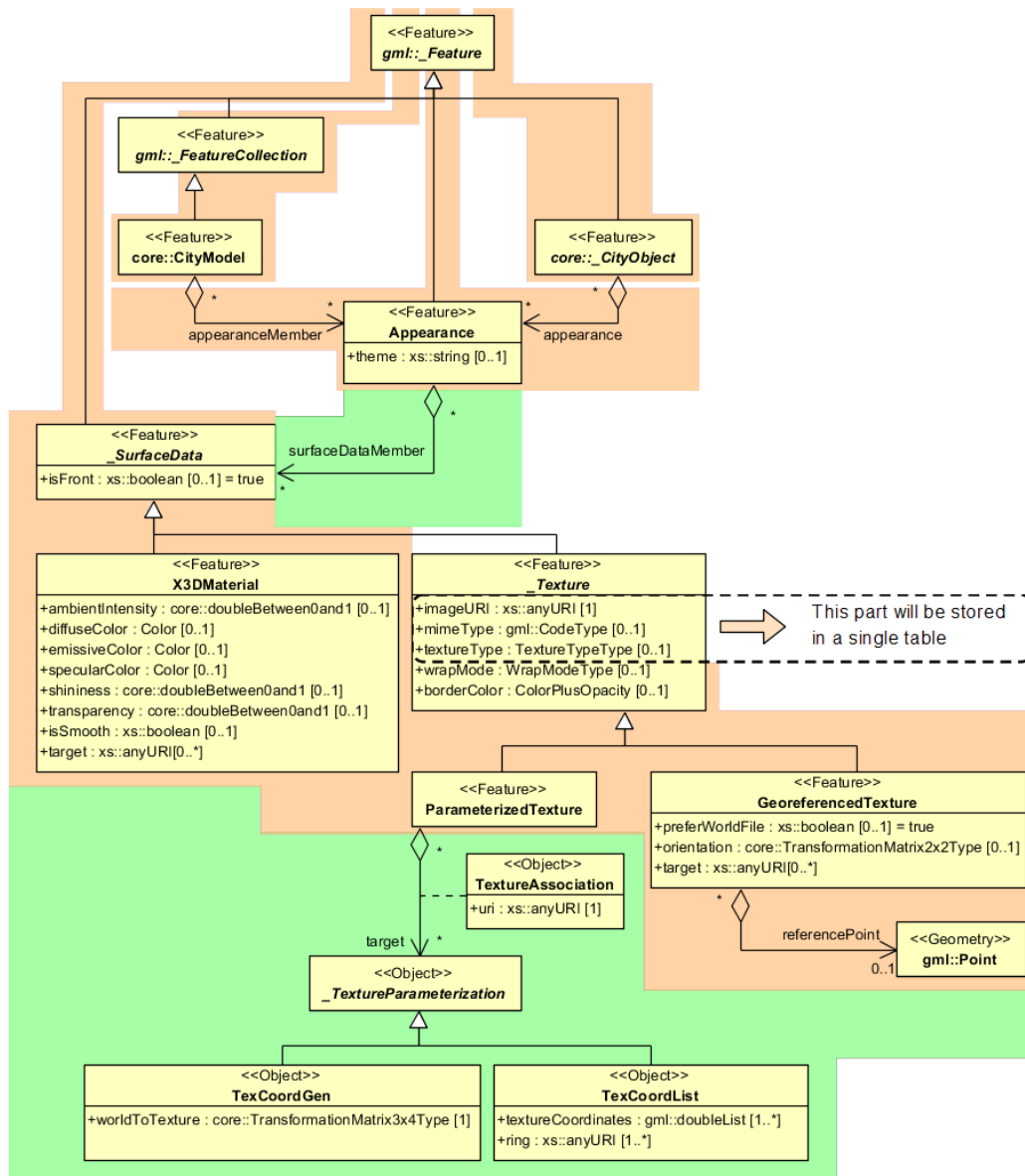


Figure 4: Appearance model

Themes are represented by an identifier only. The appearance of a city model for a given theme is defined by a set of objects of class *Appearance*, referencing this theme through the attribute *theme*. All appearance objects belonging to the same theme compose a virtual group. An *Appearance* object collects surface data relevant for a specific theme through the relation *surfaceDataMember*. Surface data is represented by objects of the abstract class *_SurfaceData*. Its only attribute is the Boolean flag *isFront*, which determines the side (front and back face of the surface) a surface data object applies to.

A constant surface property is modelled as material. A surface property, which depends on the location within the surface, is modelled as texture. Each surface object can have both a material and a texture per theme and side. This allows for providing both a constant approximation and a complex measurement of a surface's property simultaneously. If a surface object is to receive multiple textures or materials, each texture or material requires a separate theme. The mixing of themes or their usage is not explicitly defined but left to the application.

Materials define light reflection properties being constant for a whole surface object. The definition of the class *X3DMaterial* is adopted from the X3D and COLLADA specification (cf. X3D, COLLADA specification):

- *diffuseColor* defines the colour of diffusely reflected light.
- *specularColor* defines the colour of a directed reflection.
- *emissiveColor* is the colour of light generated by the surface.

All colours use RGB values with red, green, and blue channels, each defined as value between 0 and 1. Transparency is stored separately using the *transparency* element where 0 stands for fully opaque and 1 for fully transparent. *ambientIntensity* specifies the minimum percentage of *diffuseColor* that is visible regardless of light sources. *shininess* controls the sharpness of the specular highlight. 0 produces a soft glow while 1 results in a sharp highlight. *isSmooth* gives a hint for normal interpolation. If this Boolean flag is set to true, vertex normals should be used for shading (Gouraud shading). Otherwise, normals should be constant for a surface patch (flat shading). Target surfaces are specified using target elements. Each element contains the URI of one target surface geometry object.

The base class for textures is *_AbstractTexture*. Here, textures are always raster-based 2D textures. The raster image is specified by *imageURI* using a URI and may contain an arbitrary image data resource, even a preformatted request for a web service. The image data format can be defined using standard MIME types in the *mimeType* element. Textures can be qualified by the attribute *textureType*, differentiating between textures, which are specific for a certain object (*specific*) and prototypic textures being typical for that object surface (*typical*). Textures may also be classified as *unknown*. The specification of texture wrapping is adopted from the COLLADA standard. Possible values of the attribute *wrapMode* are *none*, *wrap*, *mirror*, *clamp* and *border*.

_AbstractTexture is further specialised according to the texture parameterisation, i.e. the mapping function from a location on the surface to a location in the texture image. Texture parameterisation uses the notion of texture space, where the texture image always occupies of the region $[0,1]^2$ regardless of the actual image size or aspect ratio. The lower left image corner is located at the origin. To receive textures, the mapping function must be known for each surface object.

The class *GeoreferencedTexture* describes a texture that uses a planimetric projection. Such a texture has a unique mapping function which is usually provided with the image file (e.g. georeferenced TIFF) or as a separate ESRI world file. The search order for an external georeference is determined by the Boolean flag *preferWorldFile*. Alternatively, inline specification of a georeference similar to a world file is possible. This internal georeference specification always takes precedence over any external georeference. *referencePoint* defines the location of the centre of the upper left image pixel in world space and corresponds to values 5 and 6 in an ESRI world file. Since *GeoreferencedTexture* uses a planimetric projection, *referencePoint* is two-dimensional and the *orientation* defines the rotation and scaling of the image in form of a 2x2 matrix (a list of 4 doubles in row-major order corresponding to values 1, 3, 2, and 4 in an ESRI world file). The CRS of this transformation is identical to the *referencePoint*'s CRS. If neither an internal nor an external georeference is given, the

GeoreferencedTexture is invalid. Target surfaces are specified using target elements. Each element contains the URI of one target surface geometry object. All target surface objects share the mapping function defined by the georeference.

The class *ParameterizedTexture* describes a texture with a target-dependent mapping function. Each target surface geometry object is specified as URI in the *uri* attribute of a separate *target* element. The mapping is defined by associated classes of *_TextureParameterization*:

- *TexCoordList* for the concept of texture coordinates, defining an explicit mapping of a surface's boundary points to points in texture space, and
- *TexCoordGen* when using a common 3x4 transformation matrix from world space to texture space, specified by the attribute *worldToTexture*.

2.2.4 Thematic model

The thematic model consists of the class definitions for the most important types of objects within virtual 3D city models. Most thematic classes are (transitively) derived from the basic classes *Feature* and *FeatureCollection*, the basic notions defined in ISO 19109 and GML3 for the representation of features and their aggregations. Features contain spatial as well as non-spatial attributes, which are mapped to GML3 feature properties with corresponding data types. Geometric properties are represented as associations to the geometry classes described in chapter 2.2.1 The thematic model also comprises different types of interrelationships between Feature classes like aggregations, generalizations, and associations.

The aim of the explicit modelling is to reach a high degree of semantic interoperability between different applications. By specifying the thematic concepts and their semantics along with their mapping to UML and GML3, different applications can rely on a well-defined set of *Feature* types, attributes, and data types with a standardised meaning or interpretation. In order to allow also for the exchange of objects and/or attributes that are not explicitly modelled in CityGML, the concepts of *GenericCityObjects* and *GenericAttributes* have been introduced.

2.2.4.1 Core Model

The base class of all thematic classes within CityGML's data model is the abstract class *_CityObject*. *_CityObject* provides a creation and a termination date for the management of histories of features as well as generic attributes and external references to corresponding objects in other data sets. *_CityObject* is a subclass of the GML class *Feature*, thus it may inherit multiple names from *Feature*, which may be optionally qualified by a *codeSpace*. This enables the differentiation between, for example, an official name from a popular name or names in different languages (c.f. the name property of GML objects, Cox et al., 2004). The generalisation property *generalizesTo* of *_CityObject* may be used to relate features, which represent the same real-world object in different LoD, i.e. a feature and its generalized counterpart(s). The direction of this relation is from the feature to the corresponding generalised feature.

Features of *_CityObject* and its specialized subclasses may be aggregated to a *CityModel*, which is a feature collection with optional metadata. Generally, each feature has the attributes *class*, *function*, and *usage*, unless it is stated otherwise. The *class* attribute can occur only once, while

[illegible]

Figure 5: Core Model and thematic top level classes

The subclasses of *_CityObject* comprise the different thematic fields of a city model, in the following covered by separate thematic models: building model (*_AbstractBuilding*), tunnel model (*_AbstractTunnel*), bridge model (*_AbstractBridge*), city furniture model (*CiyFurniture*), digital terrain model (*ReliefFeature*), land use model (*LandUse*), transportation model (*TransportationObject*), vegetation model (*_VegetationObject*), water bodies model (*WaterObject*) and generic city object model (*GenericCityObject*). The latter one allows for the modelling of features, which are not explicitly covered by one of the other models. The separation into these models strongly correlates with CityGML's extension modules, each defining a respective part of a virtual 3D city model.

3D objects are often derived from or have relations to objects in other databases or data sets. For example, a 3D building model may have been constructed from a two-dimensional footprint in a cadastre data set. The reference of a 3D object to its corresponding object in an external data set is essential, if an update must be propagated or if additional data is required (like the

name and address of a building's owner in a cadastral information system). In order to supply such information, each *_CityObject* may have *External References* to corresponding objects in external data sets. Such a reference denotes the external information system and the unique identifier of the object in this system.

CityObjectGroups aggregate *CityObjects* and furthermore are defined as special *CityObjects*. This implies that a group may become a member of another group realizing a recursive aggregation schema. Since *CityObjectGroup* is a feature, it has the optional attributes *class*, *function* and *usage*. The *class* attribute allows a group classification with respect to the stated function and may occur only once. The *function* attribute is intended to express the main purpose of a group, possibly to which thematic area it belongs (e.g. site, building, transportation, architecture, unknown etc.). The attribute *usage* can be used, if the object's usage differs from its function. The attributes *class*, *function* and *usage* are specified as *gml:CodeType*. The values of these properties can be enumerated in code lists.

Each member of a group may be qualified by a role name, reflecting the role each *CityObject* plays in the context of the group. Furthermore, a *CityObjectGroup* can optionally be assigned an arbitrary geometry object. This may be used to represent a generalised geometry generated from the member's geometries. The parent association linking a *CityObjectGroup* to a *CityObject* allows for the modelling of a generic hierarchical grouping concept. This concept is used, for example, to represent storeys in buildings. See Figure 5 the simplified UML diagram.

2.2.4.2 Building model

Buildings can be represented in five levels of detail (LoD0 to LoD4). The building model allows the representation of simple buildings that consist of only one component, as well as the representation of complex relations between parts of a building, e.g. a building consisting of three parts – a main house, a garage and an extension. The parts can again consist of parts etc. The subclasses *Building* and *BuildingPart* of *_AbstractBuilding* enable these modelling options. In the case of a simple, one-piece house there is only one *Building* which inherits all attributes and relations from *_AbstractBuilding* (cf. Figure 6). However, such a *Building* can also comprise *BuildingParts* which likewise inherit all properties from *_AbstractBuilding*: the building's class, function (e.g. residential, public, or industry), usage, year of construction, year of demolition, roof type, measured height, and the number and individual heights of all its storeys above and below ground (cf. Figure 7). Furthermore, *Addresses* can be assigned to *Buildings* or *BuildingParts*. In particular, *BuildingParts* may again comprise *BuildingParts* as components, because the composition relation is inherited. This way a tree-like hierarchy can be created whose root object is a *Building* and whose non-root nodes are *BuildingParts*. The attribute values are generally filled in the lower hierarchy level, because basically every part can have its own construction year and function. However, the function can also be defined in the root of the hierarchy and therefore span the whole building. The individual *BuildingParts* within a *Building* must not penetrate each other and must form a coherent object.



Figure 6: Example of buildings consisting of one and two building parts [Gröger et al., 2008]

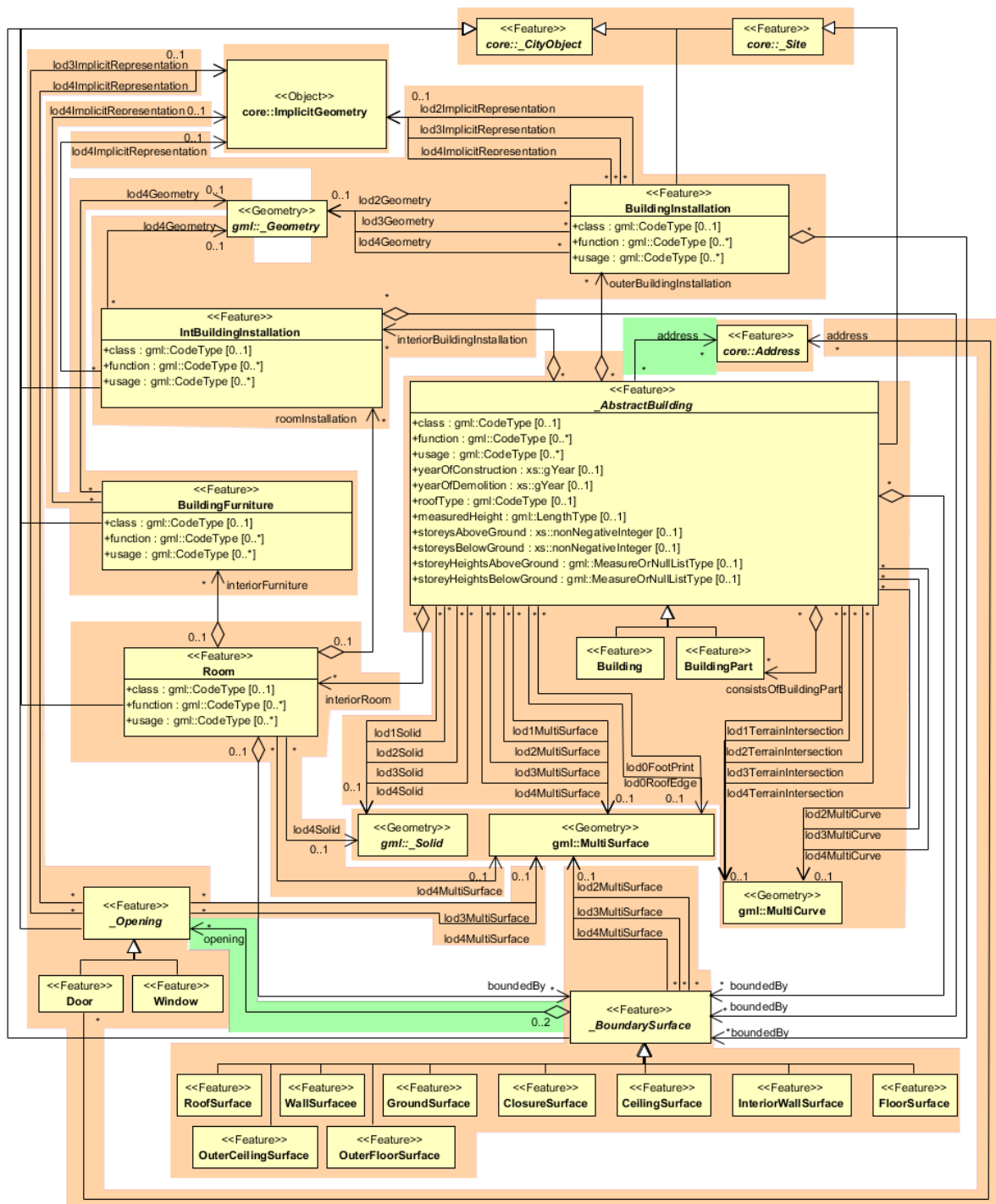


Figure 7: UML diagram of building model

The geometric representation of an *AbstractBuilding* is successively refined from LOD0 to LOD4. Therefore, a single building can have multiple spatial representations in different levels of detail at the same time by *Solid*, *MultiSurface*, and/or *MultiCurve* (cf. Figure 7).

In LoD0, the building can be represented by horizontal, 3-dimensional surfaces describing the footprint and the roof edge. In LoD1, a building model consists of a geometric representation of the building volume. Optionally, a *MultiCurve* representing the *TerrainIntersectionCurve* can be specified. This geometric representation is refined in LoD2 by additional *MultiSurface* and *MultiCurve* geometries, used for modelling architectural details like a roof overhang,

columns, or antennas. In LoD2 and higher LoDs the outer facade of a building can also be differentiated semantically by the classes *_BoundarySurface* and *BuildingInstallation*. A *_BoundarySurface* is a part of the building's exterior shell with a special function like wall (*WallSurface*), roof (*RoofSurface*), ground plate (*GroundSurface*), or closing surface (*ClosureSurface*) as shown in Figure 8. Closure surfaces can be used to virtually seal open buildings as for example hangars, allowing e.g. volume calculation. The *BuildingInstallation* class is used for building elements like balconies, chimneys, dormers, or outer stairs, strongly affecting the outer appearance of a building. A *BuildingInstallation* is used for the representation of chimneys, stairs, balconies etc. and optionally has the attributes *class*, *function*, and *usage*.

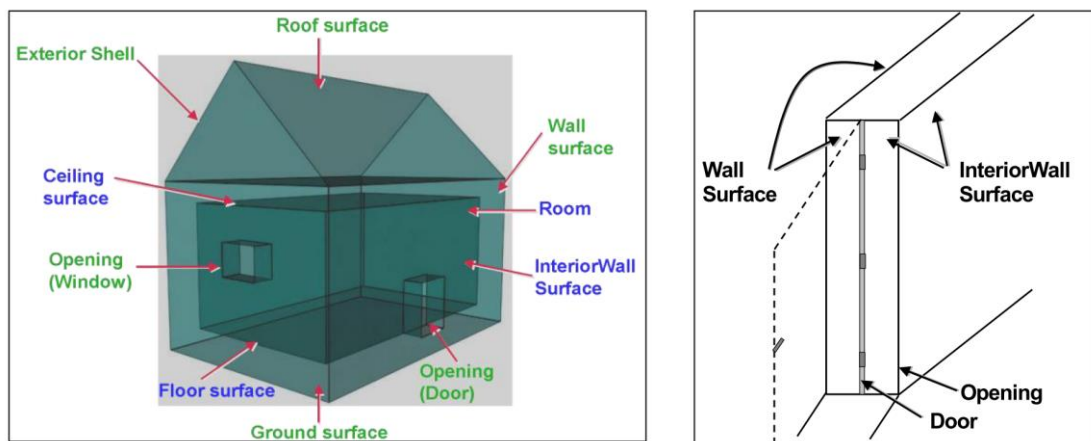


Figure 8: Boundary surfaces

In LoD3, the openings in *_BoundarySurface* objects (doors and windows) can be represented as thematic objects. In LoD4, the highest level of resolution, also the interior of a building, composed of several rooms, is represented in the building model by the class *Room*. The aggregation of rooms according to arbitrary, user-defined criteria (e.g. for defining the rooms corresponding to a certain storey) is achieved by employing the general grouping concept provided by CityGML. Interior installations of a building, i.e. objects within a building which (in contrast to furniture) cannot be moved, are represented by the class *IntBuildingInstallation*. If an installation is attached to a specific room (e.g. radiators or lamps), they are associated with the *Room* class, otherwise (e.g. in case of rafters or pipes) with *_AbstractBuilding*. A *Room* may have the attributes *class*, *function*, and *usage* referenced to external code lists. The *class* attribute allows a classification of rooms with respect to the stated function, e.g. commercial or private rooms, and occurs only once. The *function* attribute is intended to express the main purpose of the room, e.g. living room, kitchen. The attribute *usage* can be used if the object's usage differs from its function. Both attributes can occur multiple times.

The visible surface of a room is represented geometrically as a *Solid* or *MultiSurface*. Semantically, the surface can be structured into specialised *_BoundarySurfaces*, representing floor (*FloorSurface*), ceiling (*CeilingSurface*), and interior walls (*InteriorWallSurface*) (cf. Figure 8). Room furniture, like tables and chairs, can be represented in the CityGML building model with the class *BuildingFurniture*. A *BuildingFurniture* may have the attributes *class*, *function*, and *usage*.

2.2.4.3 Bridge Model

The bridge model was developed in analogy to the building model (cf. section 2.2.4.2) with regard to structure and attributes [Gröger et al., 2008]. The bridge model allows for the representation of the thematic, spatial and visual aspects of bridges and bridge parts in four levels of detail, LOD 1 – 4. A (movable or unmovable) bridge can consist of multiple *BridgeParts*. Like *Bridge*, *BridgePart* is a subclass of *_AbstractBridge* and hence, has the same attributes and relations. The relation *consistOfBridgePart* represents the aggregation hierarchy between a *Bridge* (or a *BridgePart*) and its *BridgeParts*. By this means, an aggregation hierarchy of arbitrary depth can be modelled. The semantic attributes of an *_AbstractBridge* are *class*, *function*, *usage* and *is_movable*. The attribute *class* is used to classify bridges, e.g. to distinguish different construction types (cf. Figure 9). The attribute *function* allows representing the utilization of the bridge independently of the construction. Possible values may be railway bridge, roadway bridge, pedestrian bridge, aqueduct, etc. The option to denote a usage which is divergent to one of the primary functions of the bridge (function) is given by the attribute *usage*. Each *Bridge* or *BridgePart* feature may be assigned zero or more addresses using the *address* property.

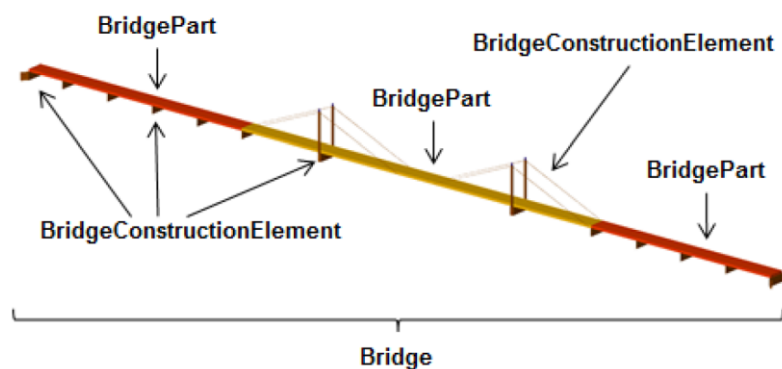


Figure 9: Example of bridge consisting of bridge parts

The spatial properties are defined by a solid for each of the four LODs (relations *lod1Solid* to *lod4Solid*). In analogy to the building model, the semantical as well as the geometrical richness increases from LOD1 (blocks model) to LOD3 (architectural model). Interior structures like rooms are dedicated to LOD4. To cover the case of bridge models where the topology does not satisfy the properties of a solid (essentially water tightness), a multi-surface representation is allowed (*lod1MultiSurface* to *lod4MultiSurface*). The line where the bridge touches the terrain surface is represented by a terrain intersection curve, which is provided for each LOD (relations *lod1TerrainIntersection* to *lod4TerrainIntersection*). In addition to the solid representation of a bridge, linear characteristics like ropes or antennas can be specified geometrically by the *lod1MultiCurve* to *lod4MultiCurve* relations.

The thematic boundary surfaces of a bridge are defined in analogy to the building module. *_BoundarySurface* is the abstract base class for several thematic classes, structuring the exterior shell of a bridge as well as the visible surfaces of rooms, bridge construction elements and both outer and interior bridge installations. From *_BoundarySurface*, the thematic classes *RoofSurface*, *WallSurface*, *GroundSurface*, *OuterCeilingSurface*, *OuterFloorSurface*, *ClosureSurface*, *FloorSurface*, *InteriorWallSurface*, and *CeilingSurface* are derived.

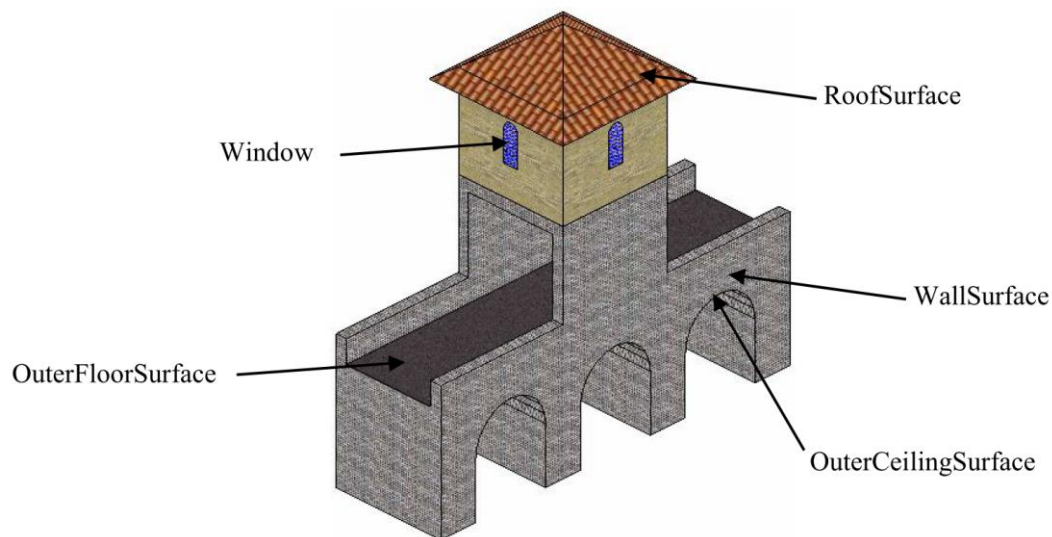


Figure 10: Different BoundarySurfaces of a bridge

Bridge elements which do not have the size, significance or meaning of a *BridgePart* can be modelled either as *BridgeConstructionElement* or as *BridgeInstallation*. Elements which are essential from a structural point of view are modelled as *BridgeConstructionElement*, for example structural elements like pylons, anchorages etc. (cf. Figure 9Figure 11). A general classification as well as the intended and actual function of the construction element are represented by the attributes *class*, *function*, and *usage*. The visible surfaces of a bridge construction element can be semantically classified using the concept of boundary surfaces representing floor (*FloorSurface*), ceiling (*CeilingSurface*), and interior walls (*InteriorWallSurface*) (cf. Figure 10). Whereas a *BridgeConstructionElement* has structural relevance, a *BridgeInstallation* represents an element of the bridge which can be eliminated without collapsing of the bridge (e.g. stairway, antenna, and railing) (cf. Figure 11). *BridgeInstallations* occur in LOD 2 to 4. The class *BridgeInstallation* contains the semantic attributes *class*, *function* and *usage*. The attribute *class* gives a classification of installations of a bridge. With the attributes *function* and *usage*, nominal and real functions of the bridge installation can be described.

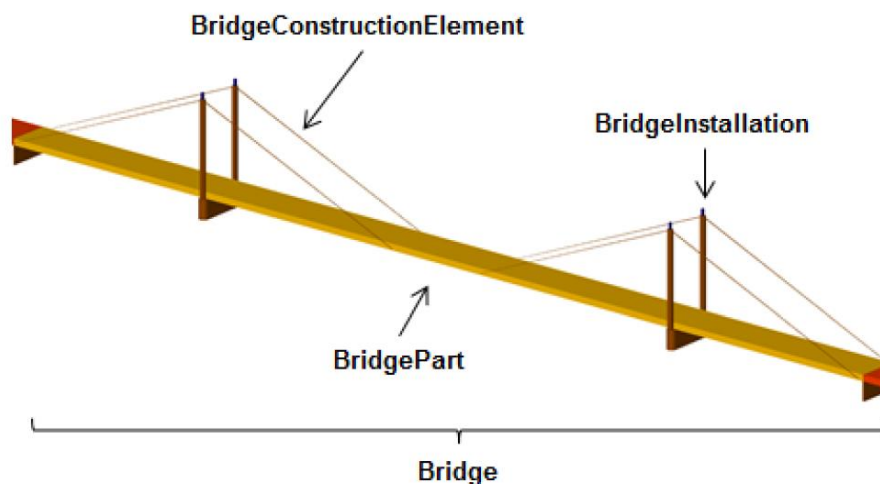


Figure 11: Example of bridge consisting of BridgeConstructionElement and BridgeInstallation

In LOD3 and LOD4, a *BoundarySurface* may contain *Openings* like doors and windows. The classes *BridgeRoom*, *IntBridgeInstallation* and *BridgeFurniture* allow for the representation of the bridge interior. They are designed in analogy to the classes *Room*, *IntBuildingInstallation* and *BuildingFurniture* of the building module and share the same meaning. The bridge interior can only be modelled in LOD4.

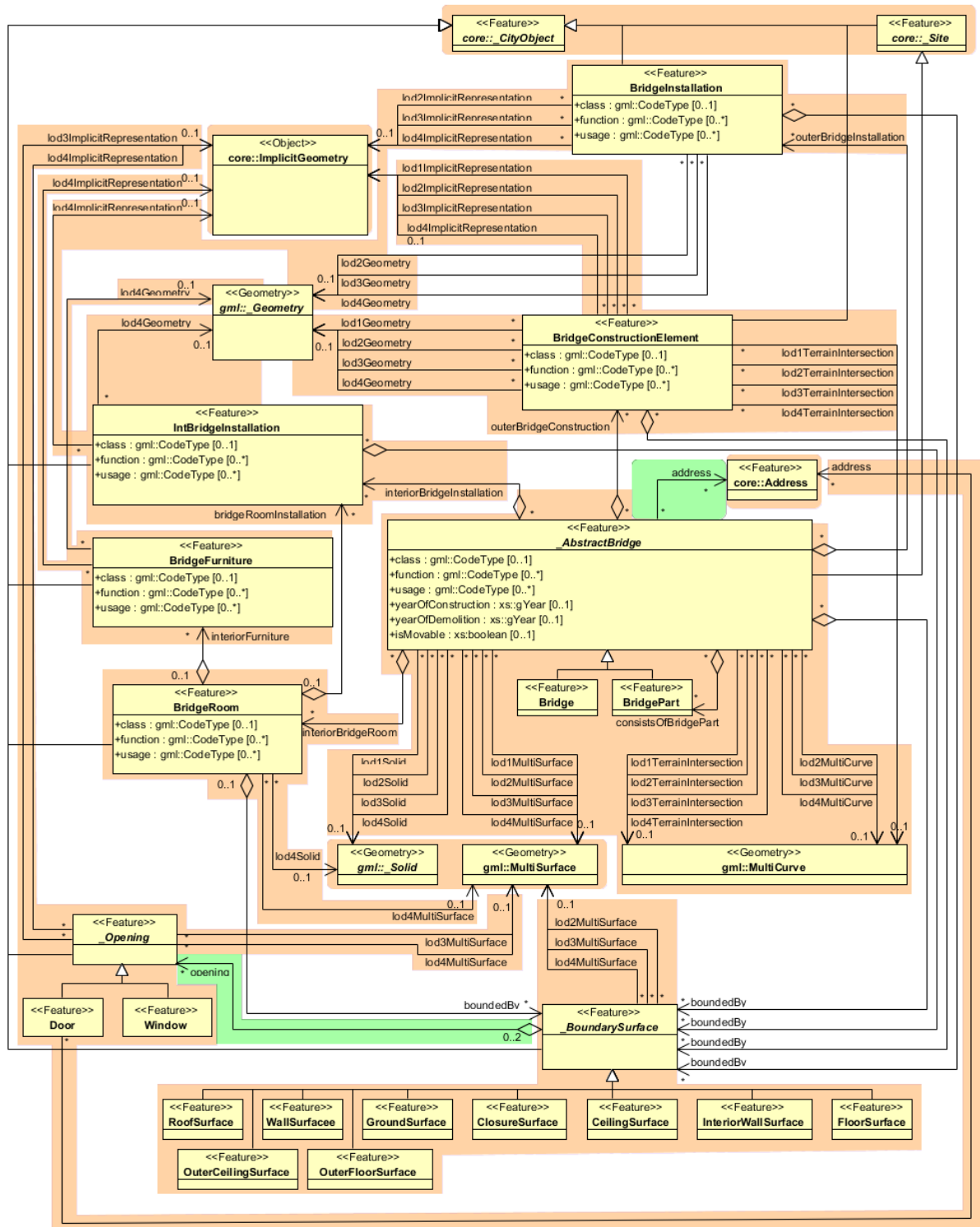


Figure 12: UML diagram of bridge model

2.2.4.4 CityFurniture Model

City furniture objects are immovable objects like lanterns, traffic lights, traffic signs, flower buckets, advertising columns, benches, delimitation stakes, or bus stops. The class CityFurniture may have the attributes *class*, *function* and *usage* (cf. UML-diagram, Figure 13). Their possible values are explained in detail in the CityGML specification. The class attribute allows an object classification like traffic light, traffic sign, delimitation stake, or garbage can, and can occur only once. The function attribute describes, to which thematic area the city furniture object belongs to (e.g. transportation, traffic regulation, architecture etc.), and can occur multiple times. The attribute *usage* denotes the real purpose of the city object, and can occur multiple times as well.

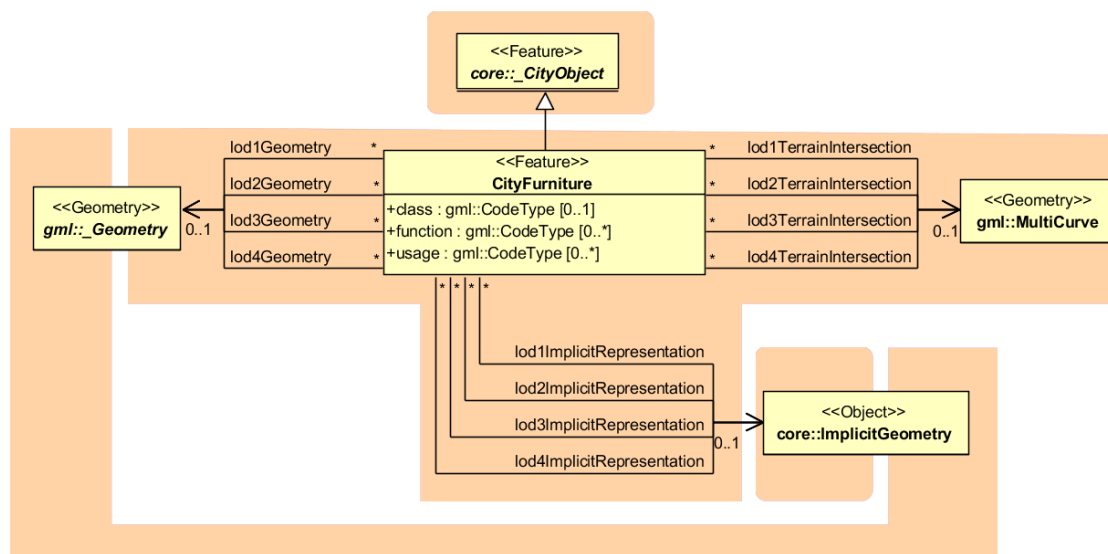


Figure 13: City furniture model

Since CityFurniture is a subclass of CityObject and hence is a feature, it inherits the attribute *gml:name*. As with any CityObject, CityFurniture objects may be assigned ExternalReferences and GenericAttributes. For ExternalReferences city furniture objects can have links to external thematic databases. Thereby, semantical information of the objects, which cannot be modelled in CityGML, can be transmitted and used in the 3D city model for further processing, for example information from systems of power lines or pipelines, traffic sign cadastre, or water resources for disaster management.

City furniture objects can be represented in city models with their specific geometry, but in most cases the same kind of object has an identical geometry. The geometry of CityFurniture objects in LoD 1-4 may be represented by an explicit geometry (lodXGeometry where X is between 1 and 4) or an *ImplicitGeometry* object (lodXImplicitRepresentation with X between 1 and 4). In the concept of ImplicitGeometry the geometry of a prototype city furniture object is stored only once in a local coordinate system and referenced by a number of features. Spatial information of city furniture objects can be taken from city maps or from public and private external information systems. In order to specify the exact intersection of the DTM with the 3D geometry of a city furniture object, the latter can have a TerrainIntersectionCurve (TIC) for each LoD. This allows for ensuring a smooth transition between the DTM and the city furniture object.

2.2.4.5 Digital terrain model

CityGML includes a very adaptable digital terrain model (DTM) which permits the combination of heterogeneous DTM types (grid, TIN, break lines, mass points) available in different levels of detail.

A DTM fitting to a certain city model is represented by the class *ReliefFeature*. This is a CityObject having the LoD step that fits the DTM as attribute. A relief consists of several *ReliefComponents*. Each of these components that are likewise CityObjects also comprises a LoD step. Individual geometrical types of the components are defined by the four subclasses of *ReliefComponent*: breaklines, triangular networks (TINs), mass points, and grids (Raster). Geometrically, the corresponding ISO 19107 or GML classes define these types: breaklines by a single *MultiCurve*, TINs by *TriangulatedSurfaces*, mass points by *MultiPoint*, and Raster by *RectifiedGridCoverage*.

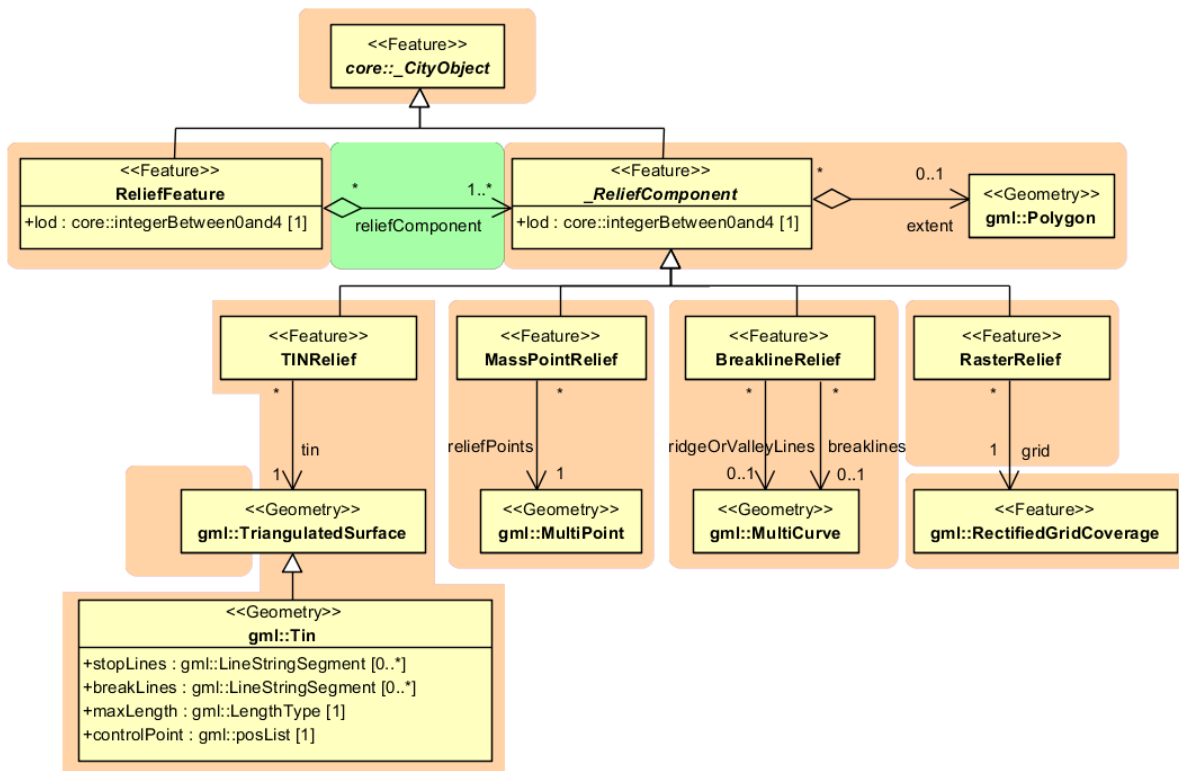


Figure 14: UML diagram representing the digital terrain model

A relief can contain *ReliefComponents* of heterogeneous type and different LoDs. A relief in LoD2, for example, can contain some LoD3-TIN-*ReliefComponents* beside a LoD2-Raster-*ReliefComponent*. In some cases even a LoD1 grid may exist in some regions of the relief.

In order to geometrically separate the individual components of a grid, which can exist in different LoD, the validity polygon of a component (*extent*) is used. This polygon defines the scope in which the component is valid. A grid with three components is shown in Figure 15. It depicts a coarse raster containing two high-resolution TINs (TIN 1 and 2). The validity polygon of the raster is represented by the blue line, while the validity polygons of the TINs are bordered in green and red. In this case, the validity polygon of the raster (grid) has two holes where the raster (grid) is not valid, although it does exist. Instead, the high-resolution TINs are used for

the representation of the terrain in these regions. That means the validity polygons of the TINs exactly fit the two holes in the validity polygon of the raster (grid).

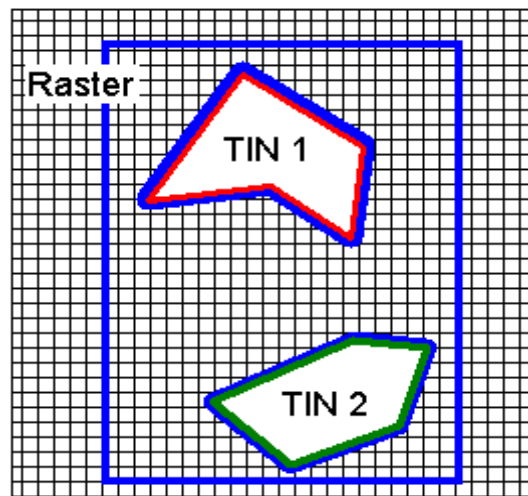


Figure 15: A relief, consisting of three components and its validity polygons
(from: [Plümer et al., 2005])

In the simplest and most frequent case, the validity polygon of a grid corresponds exactly with its Bounding box, i.e. the spatial extent of the grid.

2.2.4.6 Generic CityObject Model

The concept of generic objects and attributes is introduced to ensure the storage and exchange of 3D objects, which are not covered by explicitly modelled classes within CityGML or which requires additional attributes. These generic extensions are realised by the classes *GenericCityObject* and *GenericAttribute* (cf. Figure 16).

A *GenericCityObject* may have the attributes *class*, *function*, and *usage* are specified as *gml:CodeType*. The *class* attribute allows an object classification within the thematic area such as bridge, tunnel, pipe, power line, dam, or unknown. The *function* attribute describes to which thematic area the *GenericCityObject* belongs (e.g. site, transportation, architecture, energy supply, water supply, unknown etc.). The attribute *usage* can be used, if the object's usage differs from its function. Each *_CityObject* and all thematic subclasses can have an arbitrary number of *GenericAttributes*. Data types may be *String*, *Integer*, *Double* (floating point number), *URI* (Unified Resource Identifier), *Date*, and *gml:MeasureType*. The attribute type is defined by the selection of the particular subclass of *_genericAttribute* (*StringAttribute*, *IntAttribute* etc.). In addition, generic attributes can be grouped using the *genericAttributeSet* class which is derived from *_genericAttribute* and thus is also realized as generic attribute. Its value is the set of contained generic attributes.

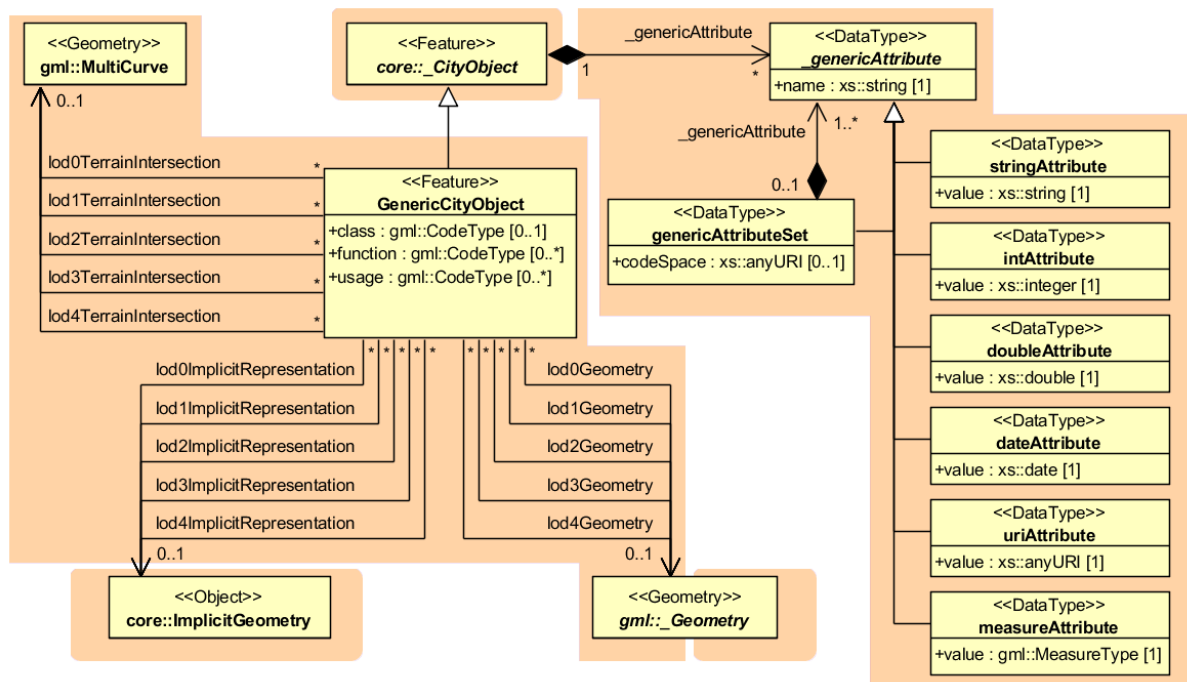


Figure 16: Generic CityObject model

The geometry of a *GenericCityObject* can either be an explicit GML3 geometry or an *ImplicitGeometry*. In the case of an explicit geometry, the object can have only one geometry for each LoD, which may be an arbitrary 3D GML geometry object (class *_Geometry*, which is the base class of all GML geometries, *lodXGeometry*, *X* in $0 \dots 4$). Absolute coordinates according to the reference system of the city model must be given for the explicit geometry. In the case of an *ImplicitGeometry*, a reference point (anchor point) of the object and optionally a transformation matrix must be given. In order to compute the actual location of the object, the transformation of the local coordinates into the reference system of the city model must be processed and the anchor point coordinates must be added. The shape of an *ImplicitGeometry* can be given as an external resource with a proprietary format, e.g. a VRML or DXF file from a local file system or an external web service. Alternatively the shape can be specified as a 3D GML3 geometry with local Cartesian coordinates using the property *relativeGeometry*.

In order to specify the exact intersection of the DTM with the 3D geometry of a *GenericCityObject*, the latter can have *TerrainIntersectionCurves* for every LoD. This is important for 3D visualization but also for certain applications like driving simulators. For example, if a city wall (e.g., the Great Wall of China) should be represented as a *GenericCityObject*, a smooth transition between the DTM and the road on the city wall would have to be ensured (in order to avoid unrealistic bumps).

2.2.4.7 LandUse Model

LandUse objects describe areas of the earth's surface dedicated to a specific land use. They can be employed to represent parcels in 3D. Figure 17 shows the UML diagram of land use objects.

Every *LandUse* object may have the attributes *class* (e.g. settlement area, industrial area, farmland etc.), *function* (purpose, e.g. cornfield), and *usage* which can be used, if the way the object is actually used differs from the function. Since the attributes *usage* and *function* may be used multiple times, storing them in only one string requires a single white space as unique separatorRelational database schema.

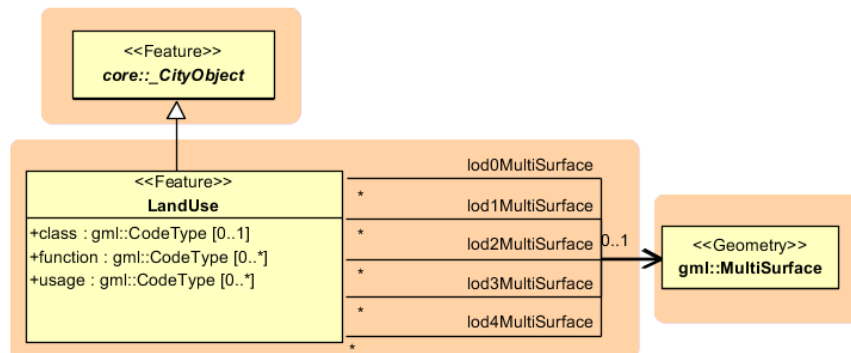


Figure 17: Landuse model

The *LandUse* object is defined for all LoD 0-4 and may have different geometries for each LoD. The surface geometry of a *LandUse* object is required to have 3D coordinate values. It must be a GML3 *MultiSurface*, which might be assigned appearance properties like material (*X3DMaterial*) and texture (*_AbstractTexture* and its subclasses).

2.2.4.8 Transportation Model

The transportation model of CityGML is a multi-functional, multi-scale model focusing on thematic and functional as well as geometrical/topological aspects. Transportation features are represented as a linear network in LoD0. Starting from LoD1, all transportation features are geometrically described by 3D surfaces.

The main class is *TransportationComplex* (cf. Figure 19) which represents, for example, a road, a track, a railway, or a square. It is composed of the parts *TrafficArea* and *AuxiliaryTrafficArea*. Figure 18 depicts an example for a LoD2 *TransportationComplex* configuration within a virtual 3D city model. The *Road* consists of several *TrafficAreas* for the sidewalks, road lanes, parking lots, and of *AuxiliaryTrafficAreas* below the raised flower beds.

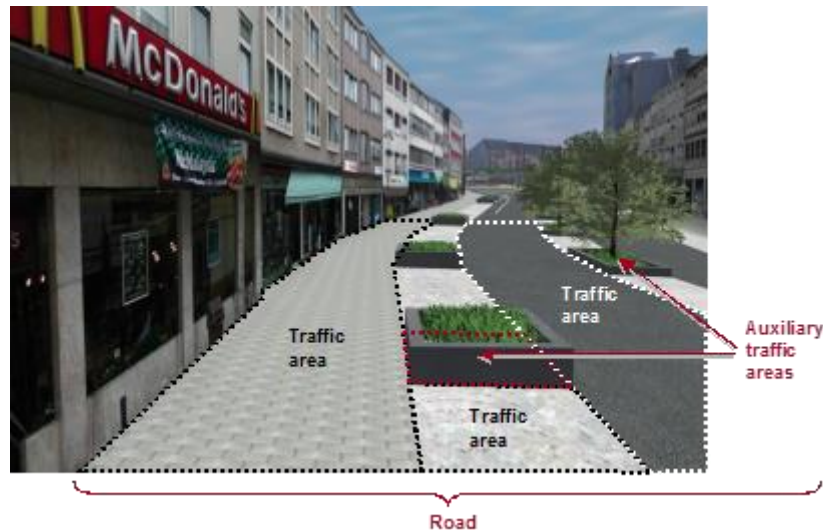


Figure 18: LoD2 representation of a transportation complex
(from: [Gröger et al., 2008])

The road itself is represented as a *TransportationComplex*, which is further subdivided into *TrafficAreas* and *AuxiliaryTrafficAreas*. The *TrafficAreas* are those elements, which are important in terms of traffic usage, like car driving lanes, pedestrian zones and cycle lanes. The *AuxiliaryTrafficAreas* are describing further elements of the road, like kerbstones, middle lanes, and green areas.

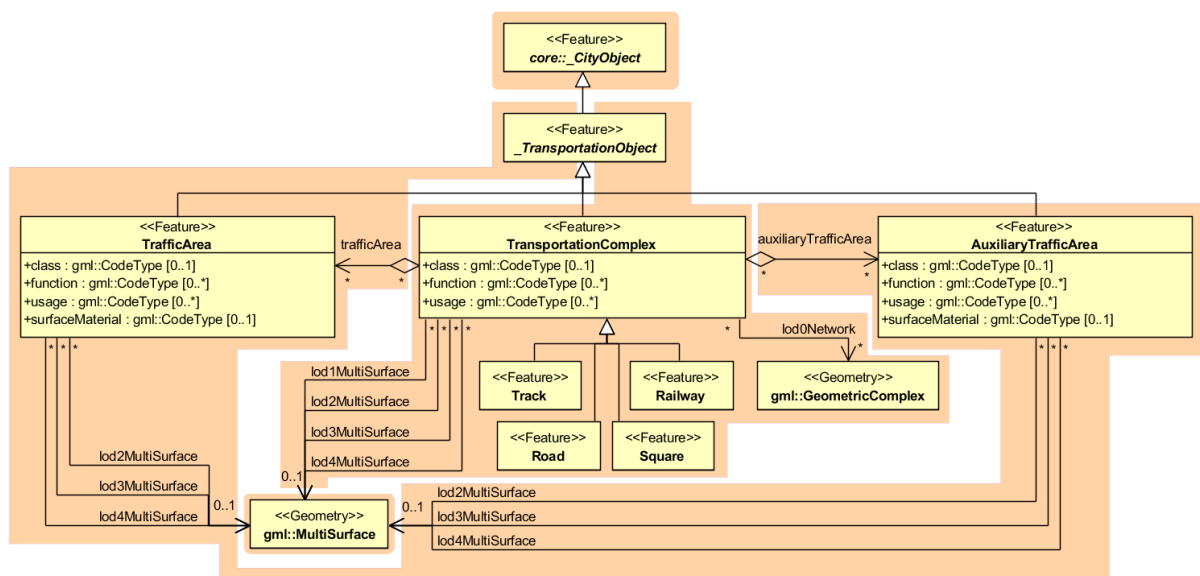


Figure 19: UML model for transportation complex

TransportationComplex objects can be thematically differentiated using the subclasses *Track*, *Road*, *Railway*, and *Square*. Every *TransportationComplex* has the attributes *class*, *function* and *usage*, referencing to the external code lists. The attribute *class* describes the classification of the object. The attribute *function* describes the purpose of the object like, for example national motorway, country road, or airport, while the attribute *usage* can be used, if the actual usage differs from the function.

In addition, both *TrafficArea* and *AuxiliaryTrafficArea* may have the attributes *class*, *function*, *usage*, and *surfaceMaterial*. The attribute *class* describe the classification of the object. For *TrafficArea*, The attribute *function* describes whether the object is a car driving lane, a pedestrian zone, or a cycle lane, while the *usage* attribute indicates which modes of transportation can use it (e.g. pedestrian, car, tram, roller skates). The attribute *surfaceMaterial* specifies the type of pavement and may also be used for *AuxiliaryTrafficAreas* (e.g. asphalt, concrete, gravel, soil, rail, grass etc.). The *function* attribute of the *AuxiliaryTrafficArea* defines, among others, kerbstones, middle lanes, or green areas. The possible values are specified in external code lists.

TransportationComplex is a subclass of *_TransportationObject* and of the root class *_CityObject*. The geometrical representation of the *TransportationComplex* varies through the different levels of detail. In the coarsest LoD0, the transportation complexes are modelled by line objects establishing a linear network. Starting from LoD1, a *TransportationComplex* provides an explicit surface geometry, reflecting the actual shape of the object, not just its centreline. In LoD2 to LoD4, it is further subdivided thematically into *TrafficAreas*, which are used by transportation, such as cars, trains, public transport, airplanes, bicycles, or pedestrians and in *AuxiliaryTrafficAreas*, which are of minor importance for transportation purposes, for example road markings, green spaces or flower tubs.

2.2.4.9 Tunnel Model

The tunnel model is closely related to the building model. It supports the representation of thematic and spatial aspects of tunnels and tunnel parts in four levels of detail, LOD1 to LOD4. The UML diagram of the tunnel model is shown in Figure 21. The pivotal class of the model is *_AbstractTunnel*, which is a subclass of the thematic class *_Site* (and transitively of the root class *_CityObject*). *_AbstractTunnel* is specialized either to a *Tunnel* or to a *TunnelPart*. Since an *_AbstractTunnel* consists of *TunnelParts*, which again are *_AbstractTunnels*, an aggregation hierarchy of arbitrary depth may be realized. Both classes *Tunnel* and *TunnelPart* inherit the attributes of *_AbstractTunnel*: the class of the tunnel, the function, the usage, the year of construction and the year of demolition. In contrast to *_AbstractBuilding*, Address features cannot be assigned to *_AbstractTunnel*.

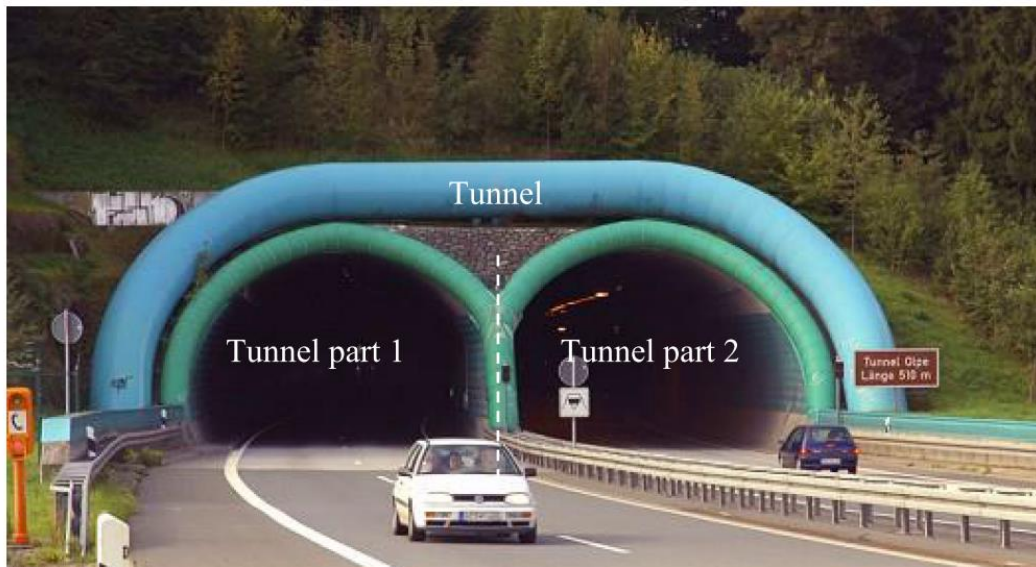


Figure 20: Example of a tunnel modelled with two tunnel parts

The geometric representation and semantic structure of an *_AbstractTunnel* is shown in Figure 21. The model is successively refined from LOD1 to LOD4. Therefore, not all components of a tunnel model are represented equally in each LOD and not all aggregation levels are allowed in each LOD. An object can be represented simultaneously in different LODs by providing distinct geometries for the corresponding LODs.

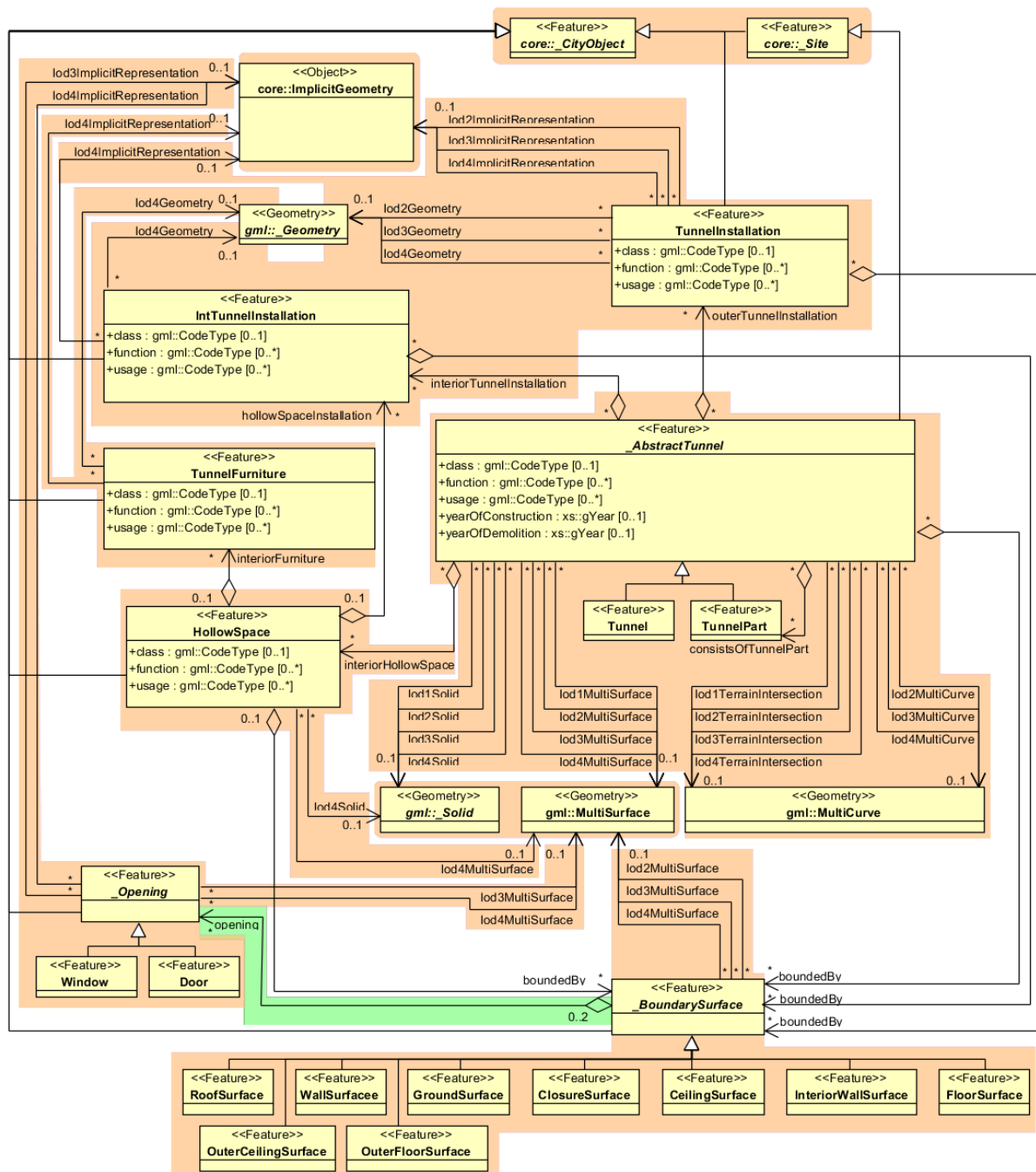


Figure 21: UML diagram of tunnel model

Similar to the building and bridge models (cf. chapters 2.2.4.2 and 2.2.4.3), only the outer shell of a tunnel is represented in LOD1 – 3, which is composed of the tunnel’s boundary surfaces to the surrounding earth, water, or outdoor air. The interior of a tunnel may only be modelled in LOD4.

In LOD1, a tunnel model consists of a geometric representation of the tunnel volume. Optionally, a *MultiCurve* representing the *TerrainIntersectionCurve* can be specified. The geometric representation is refined in LOD2 by additional *MultiSurface* and *MultiCurve* geometries. In LOD2 and higher LODs the outer structure of a tunnel can also be differentiated semantically by the classes *_BoundarySurface* and *TunnelInstallation*. A boundary surface is a part of the tunnel’s exterior shell with a special function like wall (*WallSurface*), roof

(*RoofSurface*), ground plate (*GroundSurface*), outer floor (*OuterFloorSurface*), outer ceiling (*OuterCeilingSurface*) or *ClosureSurface* (see Figure 22). The *TunnelInstallation* class is used for tunnel elements like outer stairs, strongly affecting the outer appearance of a tunnel. A *TunnelInstallation* may have the attributes *class*, *function* and *usage*.

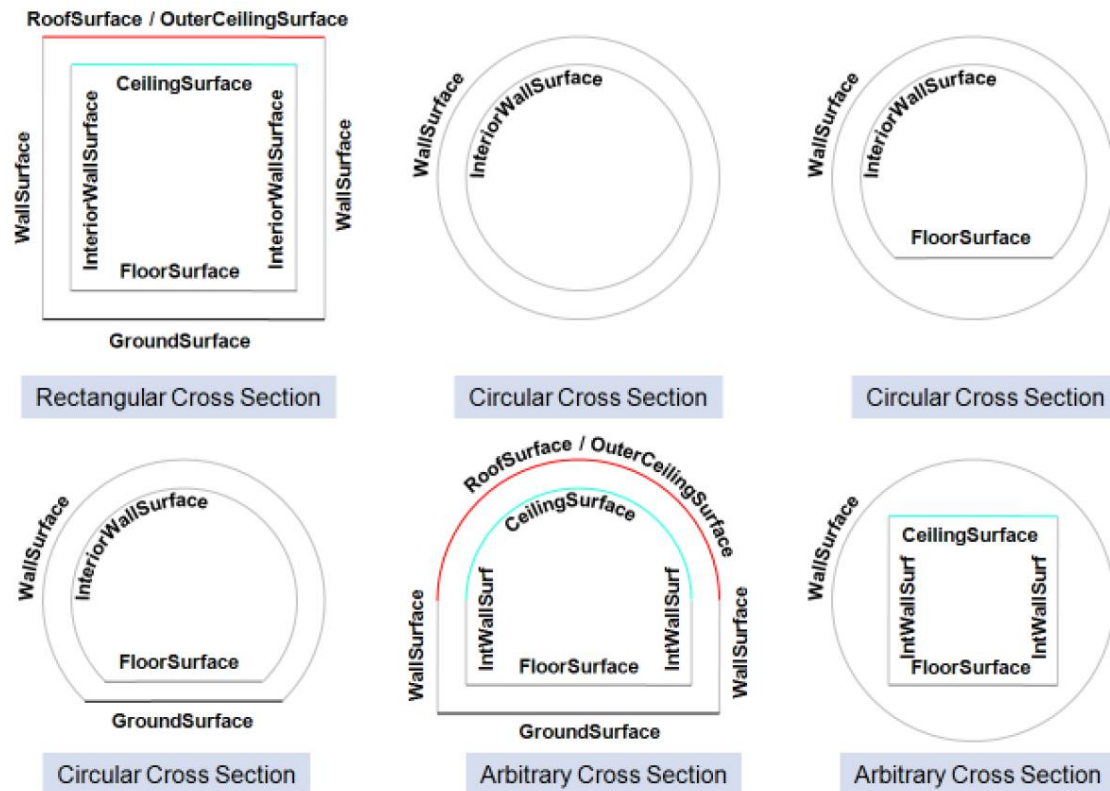


Figure 22: Different BoundarySurfaces of a tunnel

In LOD3, the openings in *_BoundarySurface* objects (doors and windows) can be represented as thematic objects. In LOD4, the highest level of resolution, also the interior of a tunnel, composed of several hollow spaces, is represented in the tunnel model by the class *HollowSpace*. This enlargement allows a virtual accessibility of tunnels, e.g. for driving through a tunnel, for simulating disaster management or for presenting the light illumination within a tunnel. The aggregation of hollow spaces according to arbitrary, user defined criteria (e.g. for defining the hollow spaces corresponding to horizontal or vertical sections) is achieved by employing the general grouping concept provided by CityGML (cf. chapter 2.2.4.1). Interior installations of a tunnel, i.e. objects within a tunnel which (in contrast to furniture) cannot be moved, are represented by the class *IntTunnelInstallation*. If an installation is attached to a specific hollow space (e.g. lamps, ventilator), they are associated with the *HollowSpace* class, otherwise (e.g. pipes) with *_AbstractTunnel*. A *HollowSpace* may have the attributes *class*, *function* and *usage* whose possible values can be enumerated in code lists. The *class* attribute allows a general classification of hollow spaces, e.g. commercial or private rooms, and occurs only once. The *function* attribute is intended to express the main purpose of the hollow space, e.g. control area, installation space, and storage space. The attribute *usage* can be used if the way the object is actually used differs from the *function*. Both attributes can occur multiple times. The visible surface of a hollow space is represented geometrically as a *Solid* or *MultiSurface*. Semantically, the surface can be structured into specialized *_BoundarySurfaces*,

representing floor (*FloorSurface*), ceiling (*CeilingSurface*), and interior walls (*InteriorWallSurface*). Hollow space furniture, like movable equipment in control areas, can be represented in the CityGML tunnel model with the class *TunnelFurniture*. A *TunnelFurniture* may have the attributes *class*, *function* and *usage*.

2.2.4.10 Vegetation Model

The vegetation model of CityGML distinguishes between solitary vegetation objects like trees and vegetation areas, which represent biotopes like forests or other plant communities. Single vegetation objects are modelled by the class *SolitaryVegetationObject*, while for areas filled with specific vegetation the class *PlantCover* is used.

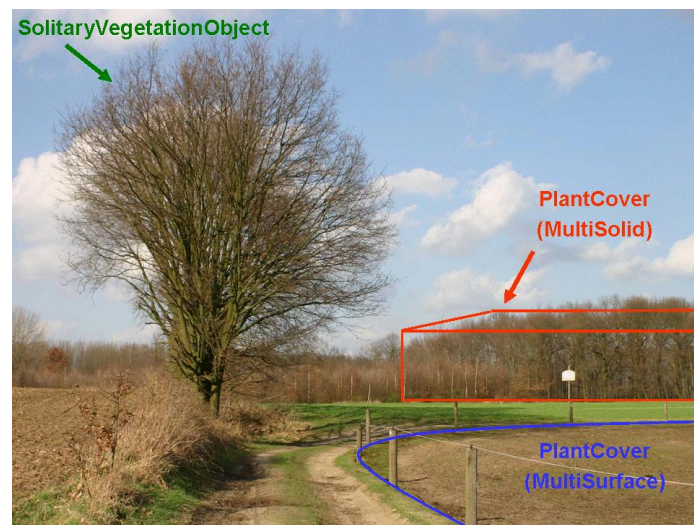


Figure 23: Image illustrates objects of the vegetation model (from: [Gröger et al., 2008])

The geometry representation of a *PlantCover* feature may be a *MultiSurface* or a *MultiSolid*, depending on the vertical extent of the vegetation. For example regarding forests, a *MultiSolid* representation might be more appropriate (cf. Figure 23).

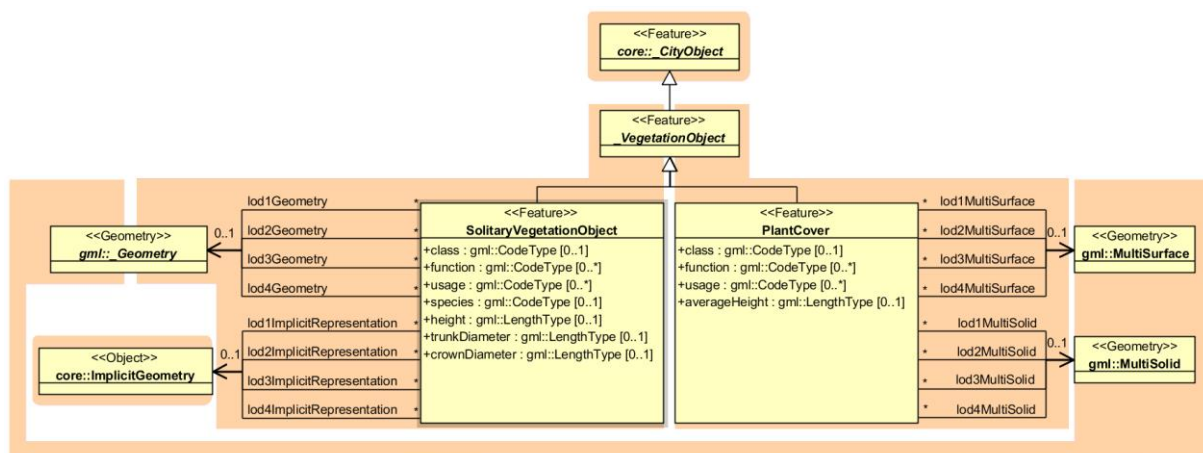


Figure 24: Vegetation Model

The UML diagram of the vegetation model is depicted in Figure 24. A *SolitaryVegetationObject* may have the attributes *class* (e.g. tree, bush, grass), *species* (species' name, e.g. *Abies alba*),

usage, and *function* (e.g. botanical monument), *height*, *trunkDiameter* and *crownDiameter*. A *PlantCover* feature may have the attributes *class* (plant community), *usage*, *function* (e.g. national forest) and *averageHeight*. Since both *SolitaryVegetationObject* and *PlantCover* are *CityObjects*, they inherit all attributes of a city object, in particular its name (*gml:name*) and an *ExternalReference* to a corresponding object in an external information system, which may contain botanical information from public environmental agencies.

The geometry of a *SolitaryVegetationObject* may be defined in LoD 1-4 by absolute coordinates, or prototypically by an *ImplicitGeometry*. Season dependent appearances may be mapped using *ImplicitGeometries*. For visualisation purposes, only the content of the library object defining the object's shape and appearance has to be swapped.

A *SolitaryVegetationObject* or a *PlantCover* may have a different geometry in each LoD. Whereas a *SolitaryVegetationObject* is associated with the *_Geometry* class representing an arbitrary GML geometry (by the relation *lodXGeometry*), a *PlantCover* is restricted to be either a *MultiSolid* or a *MultiSurface*.

2.2.4.11 WaterBodies Model

The water bodies model represents the thematic aspects and 3D geometry of rivers, canals, lakes, and basins. In LoD 2-4 water bodies are bounded by distinct thematic surfaces. These surfaces are the obligatory *WaterSurface*, defined as the boundary between water and air, the optional *WaterGroundSurface*, defined as the boundary between water and underground (e.g. DTM or floor of a 3D basin object), and zero or more *WaterClosureSurfaces*, defined as virtual boundaries between different water bodies or between water and the end of a modelled region (cf. Figure 25). A dynamic element may be the *WaterSurface* to represent temporarily changing situations of tidal flats.

Each *WaterBody* object may have the attributes *class* (e.g. lake, river, or fountain), *function* (e.g. national waterway or public swimming) and *usage* (e.g. navigable) referencing to external code lists. Since the attributes *usage* and *function* may be used multiple times, storing them in only one string requires a unique delimiter.

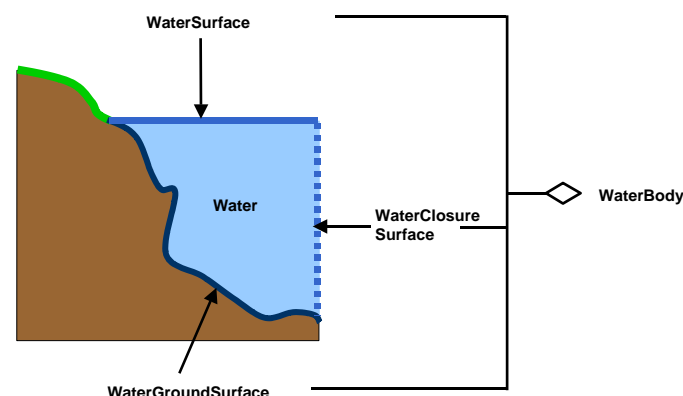


Figure 25: Definition of waterbody attributes (from: [Gröger et al., 2012])

WaterBody is a subclass of the root class *_CityObject*. The geometrical representation of the *WaterBody* varies for different levels of detail. The *WaterBody* can be differentiated semantically by the class *_WaterBoundarySurface*. A *_WaterBoundarySurface* is a part of the

water body's exterior shell with a special function like *WaterSurface*, *WaterGroundSurface* or *WaterClosureSurface*. As with any *_CityObject*, *WaterBody* objects as well as *WaterSurface*, *WaterGroundSurface*, and *WaterClosureSurface* objects may be assigned *ExternalReferences* and *GenericAttributes*.

Both LoD0 and LoD1 represent a low level of illustration and high grade of generalisation. Here the rivers are modelled as *MultiCurve* geometry and brooks are omitted. Seas, oceans, and lakes with significant extent are represented as *MultiSurfaces*. (cf. Figure 26)

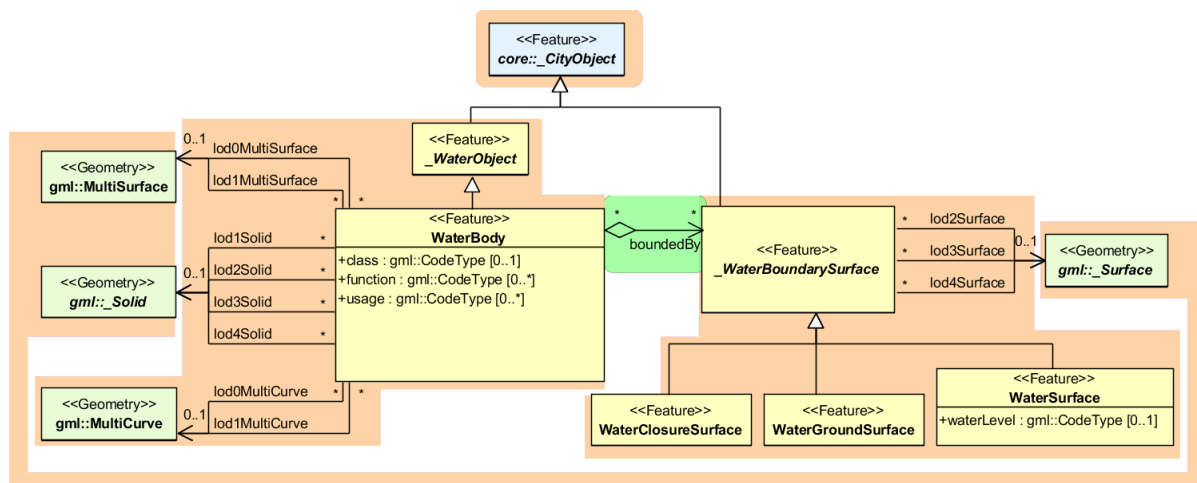


Figure 26: Waterbody model

Starting from LoD1, water bodies may also be modelled as volumes filled with water, represented by *Solids*. If a water body is represented by a *Solid* in LoD2 or higher, the surface geometries of the corresponding thematic *WaterClosureSurface*, *WaterGroundSurface*, and *WaterSurface* objects must coincide with the exterior shell of the *Solid*. This can be ensured, if for one LoD *X* the respective *lodXSurface* elements (where *X* is between 2 and 4) of *WaterClosureSurface*, *WaterGroundSurface*, and *WaterSurface* reference the corresponding polygons (using *XLink*) within the *CompositeSurface* that defines the exterior shell of the *Solid*. Furthermore, every *_WaterBoundarySurface* must have at least one associated surface geometry attached.

The water body model implicitly includes the concept of *TerrainIntersectionCurves* (TIC), e.g. to specify the exact intersection of the DTM with the 3D geometry of a *WaterBody* or to adjust a *WaterBody* or *WaterSurface* to the surrounding DTM. The rings defining the *WaterSurface* polygons implicitly delineate the intersection of the water body with the terrain or basin.

2.3 Relational database schema

2.3.1 Mapping rules, schema conventions

2.3.1.1 Mapping of classes onto tables

Generally, one or more classes of the UML diagram are mapped onto one table; the name of the table is identical to the class name (a leading underscore indicating an abstract classes is left out). Classes are combined into a single table according to the class relations as shown in the UML diagrams by using orange coloured boxes. The scalar attributes of the classes become columns of the corresponding table with identical name.

The types of the attributes are customized to corresponding database (Oracle/PostgreSQL) data types (see Table 1). Some attributes of the data type date were mapped to `TIMESTAMP WITH TIME ZONE` to allow a more accurate storage of time values.

Data type mapping (excerpt)		
UML	Oracle	PostgreSQL / PostGIS
String, anyURI	VARCHAR2, CLOB	VARCHAR, TEXT
Integer	NUMBER	NUMERIC
Double, gml:LengthType	BINARY_DOUBLE	DOUBLE PRECISION
Boolean	NUMBER(1,0)	NUMERIC
Date	DATE, TIMESTAMP WITH TIME ZONE	DATE, TIMESTAMP WITH TIME ZONE
Primitive Type (Color, TransformationMatrix, CodeType etc.)	VARCHAR2	VARCHAR
Enumeration	VARCHAR2	VARCHAR
GML Geometry, textureCoordinates	SDO_GEOMETRY	GEOMETRY
GML RectifiedGridCoverage	SDO_GEORASTER & SDO_RASTER	RASTER
Texture (only reference of type anyURI in CityGML)	BLOB	BYTEA

Table 1: Data type mapping

2.3.1.2 Explicit declaration of class affiliation

In the (meta) table `OBJECTCLASS`, all class names (attribute `CLASSNAME`) of the schema are managed. The relation of the subclass to its parent class is represented via the attribute `SUPERCLASS_ID` in the subclass as a foreign key to the ID of the parent class (see Table 2).

The table `OBJECTCLASS` is used to efficiently determine the affiliation to a class in the superclass tables. In addition, the table `CITYOBJECT` contains the attribute `OBJECTCLASS_ID` which refers to the respective table `OBJECTCLASS`. This way, while looking at a tuple in `CITYOBJECT`, the subclass and – if needed – the name of the class can be determined directly. This mechanism has also been adopted in other tables that are used to store different CityGML features, e.g. `THEMATIC_SURFACE` (for all different BoundarySurfaces of a Building feature) or `BUILDING_INSTALLATION` (outer or interior) etc.

OBJECTCLASS		
ID	CLASSNAME	SUPERCLASS_ID
0	Undefined	
1	_GML	
2	_Feature	1
3	_CityObject	2
4	LandUse	3
5	GenericCityObject	3
6	_VegetationObject	3
7	SolitaryVegetationObject	6
8	PlantCover	6
9	WaterBody	105
10	_WaterBoundarySurface	3
11	WaterSurface	10
12	WaterGroundSurface	10
13	WaterClosureSurface	10
14	ReliefFeature	3
15	_ReliefComponent	3
16	TINRelief	15
17	MassPointRelief	15
18	BreaklineRelief	15
19	RasterRelief	15
20	_Site	3
21	CityFurniture	3
22	_TransportationObject	3
23	CityObjectGroup	3
24	_AbstractBuilding	20
25	BuildingPart	24
26	Building	24
27	BuildingInstallation	3
28	IntBuildingInstallation	3
29	_BuildingBoundarySurface	3
30	BuildingCeilingSurface	29
31	InteriorBuildingWallSurface	29
32	BuildingFloorSurface	29
33	BuildingRoofSurface	29
34	BuildingWallSurface	29
35	BuildingGroundSurface	29
36	BuildingClosureSurface	29
37	_BuildingOpening	3
38	BuildingWindow	37
39	BuildingDoor	37
40	BuildingFurniture	3
41	BuildingRoom	3
42	TransportationComplex	22
43	Track	42
44	Railway	42
45	Road	42
46	Square	42
47	TrafficArea	22
48	AuxiliaryTrafficArea	22
49	FeatureCollection	2
50	Appearance	2
51	_SurfaceData	2
52	_Texture	51
53	X3DMaterial	51
54	ParameterizedTexture	52
55	GeoreferencedTexture	52

56	_TextureParametrization	1
57	CityModel	49
58	Address	2
59	ImplicitGeometry	1
60	OuterBuildingCeilingSurface	29
61	OuterBuildingFloorSurface	29
62	_AbstractBridge	20
63	BridgePart	62
64	Bridge	62
65	BridgeInstallation	3
66	IntBridgeInstallation	3
67	_BridgeBoundarySurface	3
68	BridgeCeilingSurface	67
69	InteriorBridgeWallSurface	67
70	BridgeFloorSurface	67
71	BridgeRoofSurface	67
72	BridgeWallSurface	67
73	BridgeGroundSurface	67
74	BridgeClosureSurface	67
75	OuterBridgeCeilingSurface	67
76	OuterBridgeFloorSurface	67
77	_BridgeOpening	3
78	BridgeWindow	77
79	BridgeDoor	77
80	BridgeFurniture	3
81	BridgeRoom	3
82	BridgeConstructionElement	3
83	_AbstractTunnel	20
84	TunnelPart	83
85	Tunnel	83
86	TunnelInstallation	3
87	IntTunnelInstallation	3
88	_TunnelBoundarySurface	3
89	TunnelCeilingSurface	88
90	InteriorTunnelWallSurface	88
91	TunnelFloorSurface	88
92	TunnelRoofSurface	88
93	TunnelWallSurface	88
94	TunnelGroundSurface	88
95	TunnelClosureSurface	88
96	OuterTunnelCeilingSurface	88
97	OuterTunnelFloorSurface	88
98	_TunnelOpening	3
99	TunnelWindow	98
100	TunnelDoor	98
101	TunnelFurniture	3
102	HollowSpace	3
103	TexCoordList	56
104	TexCoordGen	56
105	_WaterObject	3

Table 2: Class names

2.3.2 Database schema

In the following paragraph, the tables of the relational schema are described in detail and displayed graphically. The description is based on the remarks on UML charts in chapter 2.2. Focus is put on situations where the conversion into tables leads to changes in the model.

The figures are taken from Oracle JDeveloper, which allows to design different diagrams and reuse already defined tables. JDeveloper was used to design the database schema and extract SQL DDL scripts automatically for Oracle databases. It is a freeware IDE by Oracle and can be downloaded under: <http://www.oracle.com/technetwork/developer-tools/jdev>.

For PostgreSQL databases the open source tool pgModeler has been used to maintain the schema. Packed installers can be purchased at <http://pgmodeler.com.br/> or the user compiles the software himself from GitHub (<https://github.com/pgmodeler/pgmodeler>).

Starting from version 3.0.0 of the 3DCityDB the corresponding schema modelling projects are shipped with the release and can be edited by the user to create customized SQL scripts. However, the 3DCityDB Import/Export tool only supports the default schema, unless it is not reprogrammed against the user's new database schema.

2.3.2.1 Core Model

CITYOBJECT, CITYOBJECT_SEQ

All CityObjects (and instances of the subclasses like buildings etc.) are represented by tuples in the table CITYOBJECT. The fields are identical to the attributes of the corresponding UML class, plus additional columns for metadata like LAST_MODIFICATION_DATE, UPDATING_PERSON, REASON_FOR_UPDATE and LINEAGE.

The BoundingBox (Envelope) is realized as rectangular geometry using five points, that join the minimum and maximum x, y and z coordinates of the BoundingBox and define it completely. For backwards compatibility reasons (to Oracle 10g) the envelope cannot be stored as a volume.

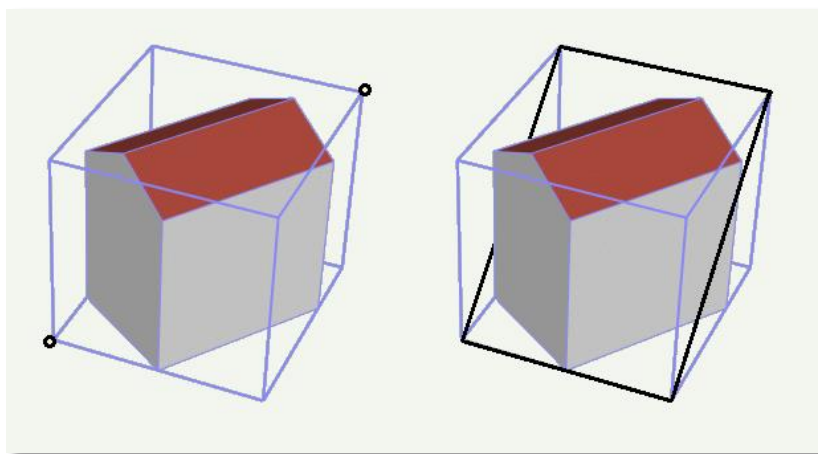


Figure 27: CityObject envelope storage as a 3D rectangle specified by (left: the two black points with minimum and maximum coordinate values respectively) and (right: black polygon using five points)

The attributes `NAME` or `NAME_CODESPACE` can contain more than one `gml:name` property. In this case they have to be separated by the string ‘`--\--`’ (more details on the following page). The CityGML exporter will then create multiple occurrences of `<gml:name>` elements.

The attribute `OBJECTCLASS_ID` provides information on the class affiliation of the `CityObject`. This helps to identify the proper subclass tables.

The next free ID value for the table `CITYOBJECT` is provided by the database sequence `CITYOBJECT_SEQ`. This ID is also reused in the separate tables for the different thematic features.

CITYMODEL, CITYMODEL_SEQ

`CityObject` features may be aggregated to a single `CityModel`. A `CityModel` serves as root element of a CityGML feature collection. In order to provide a unique identifier in table `CITYMODEL`, the next available ID value is provided by the sequence `CITYMODEL_SEQ`.

EXTERNAL_REFERENCE, EXTERNAL_REF_SEQ

The table `EXTERNAL_REFERENCE` is used to store external references; the foreign key `CITYOBJECT_ID` refers to the associated `CityObject`. The sequence `EXTERNAL_REF_SEQ` provides the next available ID value for `EXTERNAL_REFERENCE`.

CITYOBJECTGROUP, GROUP_TO_CITYOBJECT

The aggregation concept described in paragraph 2.1.1 is realized by two tables. The m:n relationship between an object group (table `CITYOBJECTGROUP`) consisting of city objects contained in `CITYOBJECT` is realized by the table `GROUP_TO_CITYOBJECT`, which associates the IDs of both tables. Table 3 shows an example, in which two buildings are grouped to a hotel complex.

CITYOBJECTGROUP (excerpt)						
ID	CLASS	CLASS_CODESPACE	FUNCTION	FUNCTION_CODESPACE	USAGE	USAGE_CODESPACE
1	NULL	NULL	Building group	NULL	Hotel	NULL

GROUP_TO_CITYOBJECT		
CITYOBJECT_ID	CITYOBJECTGROUP_ID	ROLE
2	1	Main building
4	1	Annex

CITYOBJECT (excerpt)					
ID	OBJECTCLASS_ID	GML_ID	ENVELOPE	CREATION_DATE	TERMINATION_DATE
2	26	Build1632	GEOMETRY	2015-02-02 09:26:07.441+01	NULL
4	26	Build1633	GEOMETRY	2015-02-02 09:26:07.441+01	NULL
1	23	Group1700	NULL	2015-02-02 09:26:07.441+01	NULL

Table 3: Cityobjectgroup tables

For attributes *class*, *function* and *usage* there is an additional `_CODESPACE` column in order to specify the source of code lists used for values (e.g. by a globally unique URL). As a

CityGML feature like *CityObjectGroup* can have multiple instances of attributes *class*, *function* and *usage* but only one target column exist in the table, values are separated by the string sequence ‘--/\--’. The CityGML exporter will then create multiple occurrences of corresponding elements. Normalization rules were not applied in this case in order to avoid many joins when querying all information of building objects. Array types weren’t used either as their implementation varies between different database systems.

This concept applies to all CityGML features and can therefore be found in every object table (except for boundary surfaces of buildings, bridges and tunnels). They do not appear once in the CITYOBJECT table, because they are belonging to the namespace of a certain thematic module and should be stored along with other attributes of that feature.

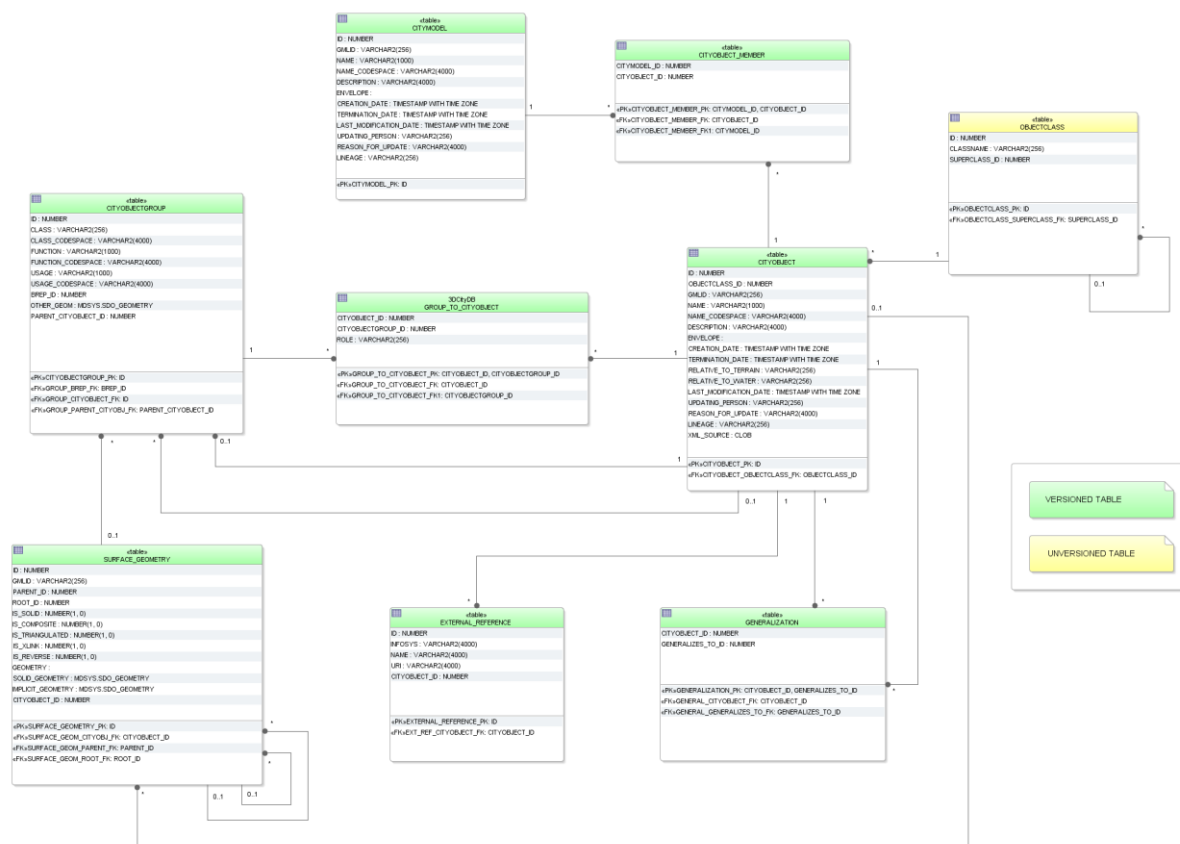


Figure 28: Database schema of the CityGML core elements

2.3.2.2 Tables for geometry representation

The representation of the geometry stored in table **SURFACE_GEOMETRY** differs substantially from the UML chart explained in the CityGML specification; nevertheless it offers about the same functionality:

SURFACE_GEOMETRY, SURFACE_GEOMETRY_SEQ

In the database schema the geometry consists of planar surfaces which correspond each to one entry in the table **SURFACE_GEOMETRY**. The surface-based geometry is stored as attribute **GEOMETRY** (in each case exactly one planar polygon, possibly including holes). The implicit geometry is stored as attribute **IMPLICIT_GEOMETRY**. The volumetric geometry is stored as

attribute `SOLID_GEOMETRY` and its boundary surfaces (outer shell) will be stored as attribute `GEOMETRY` as well. Any surface may have textures or a colour on both sides. Textures are stored within the tables which implement the appearance model (cf. chapter 2.2.3).

The geometry information in the fields `GEOMETRY` and `IMPLICIT_GEOEMTRY` of the table `SURFACE_GEOMETRY` is limited as follows:

Geometry storage in Surface Geometry - polygonal geometry	
Oracle	PostGIS
<ul style="list-style-type: none"> • <code>SDO_GTYPE</code> must have the type <i>Polygon</i>, i.e. a polygon with 3D coordinates (<code>SDO_GTYPE</code> = 3003), • <code>SDO_ETYPE</code> must be 1003/2003 with <code>SDO_INTERPRETATION</code> = 1 (i.e. polygon with 3D coordinates in the boundary, bounded just by line segments, possibly including holes) • In addition Oracle allows the representation of a rectangle by two corner points (<code>SDO_ETYPE</code>=1003/2003, with <code>SDO_INTERPRETATION</code> = 3) • <code>SDO_SRID</code> of implicit geometries can be any SRID Oracle supports. No spatial index is defined on the column by default. 	<ul style="list-style-type: none"> • Only POLYGON Z is allowed, i.e. a polygon with 3D coordinates • Polygons might have holes • The <code>IMPLICIT_GEOMETRY</code> column has no SRID defined. Thus, entries in that column will have the SRID 0 automatically

Table 4: Storage of polygonal geometry

A solid is the basis for 3-dimensional geometry. The extent of a solid is defined by the boundary surfaces (outer shell). A shell is represented by a composite surface, where every shell is used to represent a single connected component of the boundary of a solid. It consists of a composite surface (a list of orientable surfaces) connected in a topological cycle. Unlike a ring, a shell's elements have no natural sort order. Like rings, shells are simple. The geometry in the field `SOLID_GEOMETRY` of the table `SURFACE_GEOMETRY` is limited as follows:

Geometry storage in Surface Geometry - 3D geometry	
Oracle	PostGIS
<ul style="list-style-type: none"> • SDO_GTYPE must have the type <i>Solid</i>, i.e. a solid with 3D coordinates (SDO_GTYPE = 3008) • SDO_ETYPE must be 1007 (simple solid) or 1008 (composite solid). • A simple solid can be represented by using several polygons as its boundary (SDO_ETYPE=1007, with SDO_INTERPRETATION = 1). • The composite solid can be constructed with a number of simple solids, e.g. a composite solid with 4 simple solids (SDO_ETYPE=1008, with SDO_INTERPRETATION = 4) 	<ul style="list-style-type: none"> • Only POLYHEDRALSURFACE is allowed, i.e. the outer shell of a solid with 3D coordinates • A simple polyhedral surface can be represented by using several polygons as its boundary

Table 5: Storage of 3D geometry

Surfaces can be aggregated to form a complex of surfaces or the boundary of a volumetric object. The aggregation of multiple surfaces, e.g. F_1 to F_n , (IDs 6 to 10 in Figure 29 / Figure 30) is realized the way that the newly created surface tuple F_{n+1} (ID 2) is not assigned a geometry (cf. Table 6). Instead, the PARENT_ID of the surfaces F_1 to F_n refer to the ID of F_{n+1} .

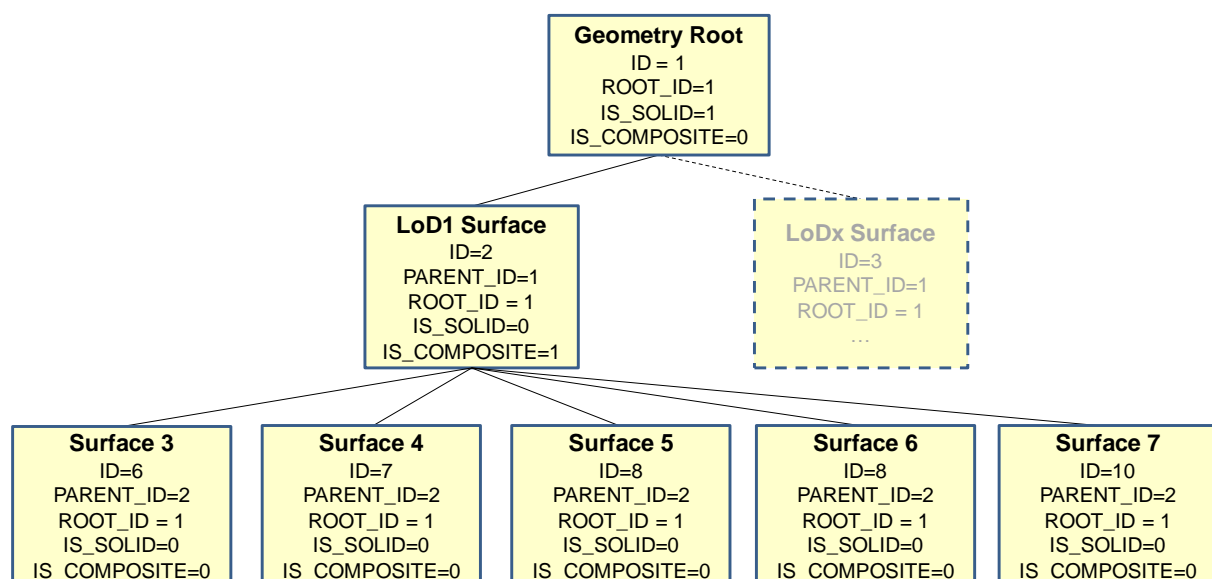


Figure 29: Geometry hierarchy refers to solid geometry shown in Figure 30

In addition, a further tuple (ID 1) is introduced, which represent the solid and defines the root element of the whole aggregation structure. Each surface references to its root, using the ROOT_ID attribute. This information has big influence on the system performance, as it allows to avoid recursive queries. If e.g. the retrieval of all surface elements forming a specific building is of importance, simply those tuples have to be selected which contain the related ROOT_ID. On the downside there also follows the limitation that each tuple in SURFACE_GEOMETRY can only belong to one aggregate.

Various flags characterise the type of aggregation: `IS_TRIANGULATED` denotes a `TriangulatedSurface`, `IS_SOLID` distinguishes between surface (0) and solid (1), and `IS_COMPOSITE` defines whether this is an aggregate (e.g. `MultiSolid`, `MultiSurface`) or a composite (e.g., `CompositeSolid`, `CompositeSurface`).

Based on these flags the geometry types listed in 5 can be distinguished. To distinguish a `MultiSolid` from a `MultiSurface` its child elements have to be analysed: In case the child is a `Solid`, the geometry can be identified as `MultiSolid`.

	isSolid	isComposite	isTriangulated	Geometry	SOLID_ GEOMETRY
Polygon, Triangle, Rectangle	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	GEOMETRY	NULL
MultiSurface	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL	NULL
CompositeSurface	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	NULL
TriangulatedSurface	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	NULL	NULL
Solid	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL	GEOMETRY
MultiSolid	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL	NULL
CompositeSolid	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL	GEOMETRY

Table 6: Attributes determining aggregation types

Aggregated surfaces can be grouped again with other (compound) surfaces, by generating a common parent. This way, arbitrary aggregations of `Surfaces`, `CompositeSurfaces`, `Solids`, `CompositeSolids` can be formed. Since all tuples in an aggregated geometry refer to the same `ROOT_ID` all tuples can be retrieved efficiently from the table by selecting those tuples with the same `ROOT_ID`.

The aggregation schema allows for the definition of nested aggregations (hierarchy of components). For example, a building geometry (*CompositeSolid*) can be composed of the house geometry (*CompositeSolid*) and the garage geometry (*Solid*), while the house's geometry is further decomposed into the roof geometry (*Solid*) and the geometry of the house body (*Solid*).

In addition, the foreign key `CITYOBJECT_ID` refers directly to the CityGML features to which the geometry belongs. In order to select all geometries forming the city object one only has to select those with the same `CITYOBJECT_ID`.

In order to provide a unique identifier in table `SURFACE_GEOMETRY`, the next available ID value is provided by the sequence `SURFACE_GEOMETRY_SEQ`.

Example: The geometry shown in the figure below consists of seven surfaces which delimit a volumetric object. In the table it is represented by the following rows:

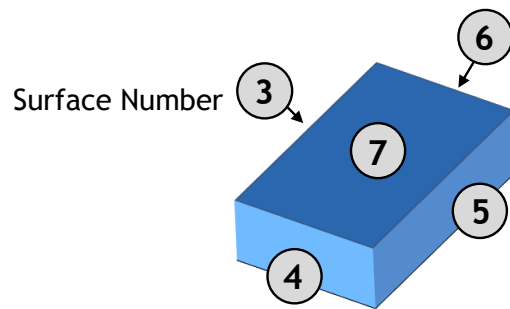


Figure 30: LoD 1 building - closed volume bounded by a CompositeSurface which consists of single polygons

SURFACE_GEOMETRY							
ID	GMLID	PARENT_ID	ROOT_ID	IS_SOLID	IS_COMPOSITE	GEOMETRY	SOLID_GEOMETRY
1	UUID_lod1	NULL	1	1	0	NULL	GEOENTRY for Solid
2	lod1Surface	1	1	0	1	NULL	NULL
6	Left1	2	1	0	0	GEOMETRY for surface 3	NULL
7	Front1	2	1	0	0	GEOMETRY for surface 4	NULL
8	Right1	2	1	0	0	GEOMETRY for surface 5	NULL
9	Back1	2	1	0	0	GEOMETRY for surface 6	NULL
10	Roof1	2	1	0	0	GEOMETRY for surface 7	NULL

Table 7: Excerpt of table SURFACE_GEOMETRY representing the example given in **Figure 30**

In addition, two further attributes are included in SURFACE_GEOMETRY: IS_XLINK and IS_REVERSE.

IS_XLINK

CityGML allows for sharing of geometry objects between different geometries or different thematic features using the XLink concept of GML3. For this purpose, the geometry object to be shared is assigned an unique *gml:id* which may be referenced by a GML geometry property element through its *xlink:href* attribute. This concept allows for avoiding data redundancy. Furthermore, CityGML does not employ the built-in topology package of GML3 but rather uses the XLink concept for the explicit modelling of topology (see [Gröger et al. 2008], p. 25).

Although an XLink can be seen as a pointer to an existing geometry object the SURFACE_GEOMETRY table does not offer a foreign key attribute which could be used to refer to another tuple within this table. The main reason for this is that the referenced tuple typically belongs to a different geometry aggregate, e.g. a different *gml:Solid* object, and thus contains different values for its ROOT_ID and PARENT_ID attributes. Therefore, foreign keys would violate the aggregation mechanism of the SURFACE_GEOMETRY table.

The recommended way of resolving of XLink references to geometry objects requires two steps: First, the referenced tuple of the SURFACE_GEOMETRY table has to be identified by searching the GMLID column for the referenced *gml:id* value. Second, all attribute values of

the identified tuple have to be copied to a new tuple. However, the ROOT_ID and PARENT_ID of this new tuple have to be set according to the context of the referencing geometry property element.

Please note:

1. If the referenced tuple is the top of an aggregation (sub)hierarchy within the SURFACE_GEOMETRY table then also **all nested tuples have to be recursively copied** and their ROOT_ID and PARENT_ID have to be adapted.
2. Copying existing entries of the SURFACE_GEOMETRY table results in tuples sharing the same GMLID. Thus, these values cannot be used as a primary key.

When it comes to exporting data to a CityGML instance document, XLink references can be rebuilt by keeping track of the GMLID values of exported geometry tuples. Generally, for *each and every* tuple to be exported it has to be checked whether a geometry object with the same GMLID value has already been processed. If so, the export routine should make use of an XLink reference.

However, checking the GMLID of each and every tuple may dramatically slow down the export process. For this reason, the IS_XLINK flag of the SURFACE_GEOMETRY has been introduced. It may be used to explicitly mark just those tuples for which a corresponding check has to be performed. The IS_XLINK flag should be used in the following manner. The Importer/Exporter provides a corresponding reference implementation.

1. During import

- a. By default, the IS_XLINK flag is set to “0”.
- b. If existing tuples have to be copied due to an XLink reference, IS_XLINK has to be set to “1” for *each and every* copy. Please note, that this rule comprises all copies of nested tuples.
- c. Furthermore, IS_XLINK has to be set to “1” on the original tuple addressed by the XLink reference. If this tuple is the top of an aggregation (sub)hierarchy, IS_XLINK remains “0” for all nested tuples.

2. During export

- a. The export process just has to keep track of the GMLID values of those geometry tuples where IS_XLINK is set to “1”.
- b. When it comes to exporting a tuple with IS_XLINK set to “1”, the export process has to check whether it already came across the same GMLID and, thus, can make use of an XLink reference in the instance document.
- c. For each tuple with IS_XLINK=0 no further action has to be taken.

Especially due to (2c), the IS_XLINK attribute helps to significantly speed up the export process when rebuilding XLink references. Please note, that this is the only intended purpose of the IS_XLINK flag.

IS_REVERSE

The IS_REVERSE flag is used in the context of gml:OrientableSurface geometry objects. Generally, an OrientableSurface instance cannot be represented within the SURFACE_GEOMETRY table since it cannot be encoded using the flags IS_SOLID, IS_COMPOSITE, and IS_TRIANGULATED (cf. Table 6). However, the IS_REVERSE flag is used to encode the information provided by an OrientableSurface and to rebuild OrientableSurfaces during data export.

According to GML3, an OrientableSurface consists of a base surface and an orientation. If the orientation is “+”, then the OrientableSurface is identical to the base surface. If the orientation is “-“, then the OrientableSurface is a reference to a surface with an up-normal that reverses the direction for this OrientableSurface.

During import, only the base surfaces are written to the SURFACE_GEOMETRY table. The following rules have to be obeyed in the context of OrientableSurface:

1. If the orientation of the OrientableSurface is “-“, then
 - a. The direction of the base surface has to be reversed prior to importing it (generally, this means reversing the order of coordinate tuples).
 - b. The IS_REVERSE flag has to be set to “1” for the corresponding entry in the SURFACE_GEOMETRY table.
 - c. If the base surface is an aggregate, then steps (a) and (b) have to be recursively applied for all of its surface members.
2. If the OrientableSurface is identical to its base surface (i.e., if its orientation is “+”), then the base surface can be written to the SURFACE_GEOMETRY table without taking any further action. The IS_REVERSE flag has to be set to “0” (which is also the *default value*).
3. Please note, that it is not sufficient to just rely on the *gml:orientation* attribute of an OrientableSurface in order to determine its orientation since OrientableSurfaces may be arbitrarily nested.

Flipping the direction of the base surface in step (1a) is essential in order to guarantee that the SDO_GEOMETRY objects stored within the GEOMETRY column are always correctly oriented. This enables applications to just access the GEOMETRY column without having to interpret further attributes of the SURFACE_GEOMETRY table. For example, in the case of a viewer application this allows for a fast rendering of a virtual 3d city scene.

When exporting CityGML instance documents, the IS_REVERSE flag can be used to rebuild OrientableSurface in the following way:

1. If the IS_REVERSE flag is set to “1” for a table entry, the exporter routine has to reverse the direction of the corresponding surface object prior to exporting it (again, this means reversing the order of coordinate tuples).
2. The surface object has to be wrapped by a gml:OrientableSurface object with *gml:orientation*="-".

3. If the surface object is an aggregate, its surface members having the *same value* for the IS_REVERSE flag *may not* be embraced by another OrientableSurface. However, if the IS_REVERSE value changes, e.g., from “1” for the aggregate to “0” for the surface member, also the surface member has to be embraced by a gml:OrientableSurface according to (2). Since there might be nested structures of arbitrary depth this third rule has to be applied recursively.

Like with the IS_XLINK flag, the Importer/Exporter tool provides a reference implementation of the IS_REVERSE flag.

2.3.2.3 Appearance Model

APPEARANCE, APPEARANCE_SEQ

The table APPEARANCE contains information about the surface data of objects (attribute DESCRIPTION), its category is stored in attribute THEME. Since each city model or city object may store its own appearance data, the table APPEARANCE is related to the tables for the base classes CityObject and CityModel by two foreign keys which may be used alternatively. The classes *Appearance* and *_SurfaceData* represent features, which can be referenced by GML identifiers. For this reason the attributes GMLID and were added to the corresponding tables.

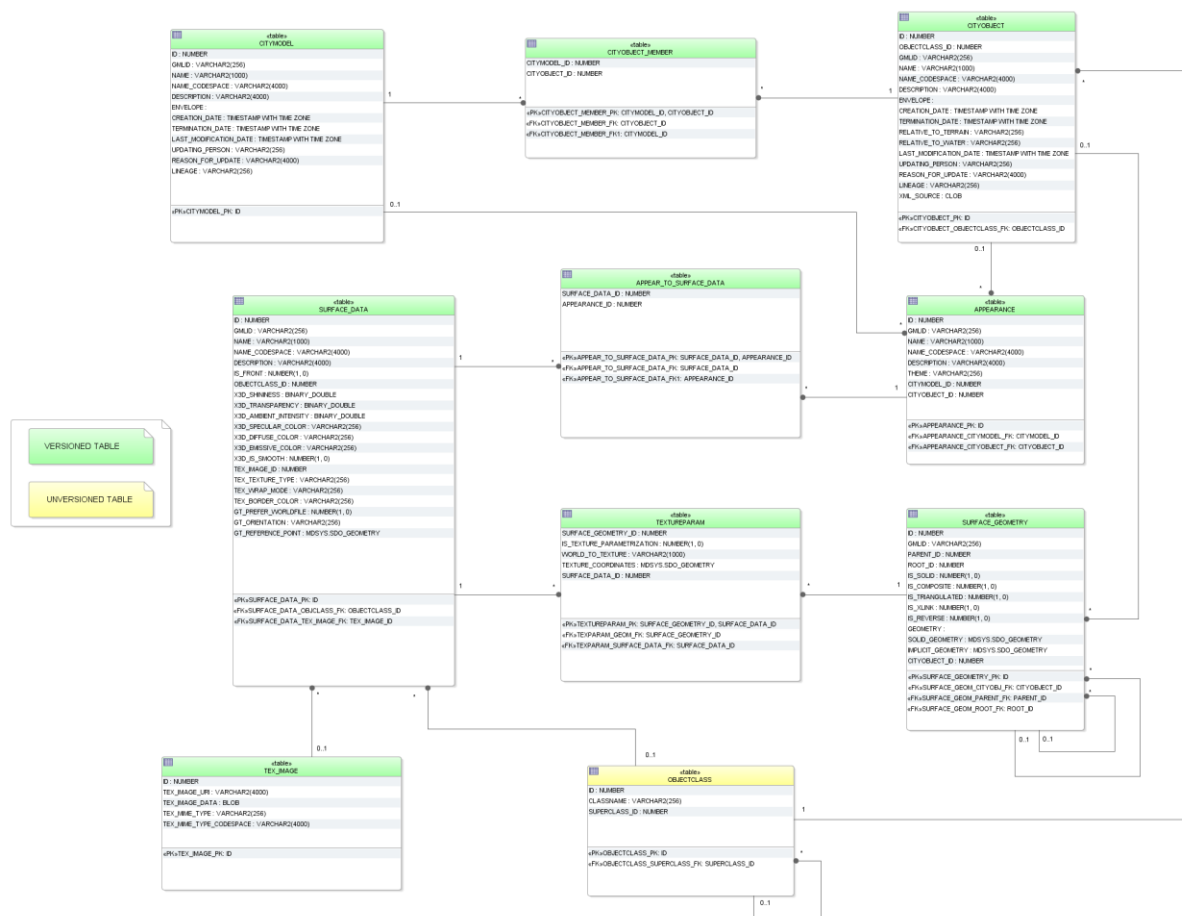


Figure 31: Appearance database schema

SURFACE_DATA, TEX_IMAGE, APPEAR_TO_SURFACE_DATA

An appearance is composed of data for each surface geometry object. Information on the data types and its appearance are stored in table SURFACE_DATA.

IS_FRONT determines the side a surface data object applies to (IS_FRONT=1: front face IS_FRONT=0: back face of a surface data object). The character string TYPE denotes if materials or textures are used for the specific object (values: *X3DMaterial*, *Texture* or *GeoreferencedTexture*). Materials are specified by the attributes X3D_xxx which define its graphic representation. Details on using georeferenced textures, such as orientation and reference point, are contained in attributes GT_xxx. See chapter 2.2.3 for more information on SURFACE_DATA attributes or the CityGML specification [Gröger et al. 2012, p. 33-45] which explains the texture mapping process in detail.

Raster-based 2D textures are stored in table TEX_IMAGE. The name of the corresponding images for example is specified by the attribute TEX_IMAGE_URI. The texture image can be stored within this table in the attribute TEX_IMAGE_DATA using the BLOB data type under Oracle and the BYTEA data type under PostgreSQL.

Table APPEAR_TO_SURFACE_DATA represents the interrelationship between appearances and surfaces for different themes.

TEXTUREPARAM

Attributes for mapping textures to objects (point list or transformation matrix) which are defined by the CityGML classes *_TextureParameterization*, *TexCoordList*, and *TexCoordGen* are stored in the table TEXTUREPARAM.

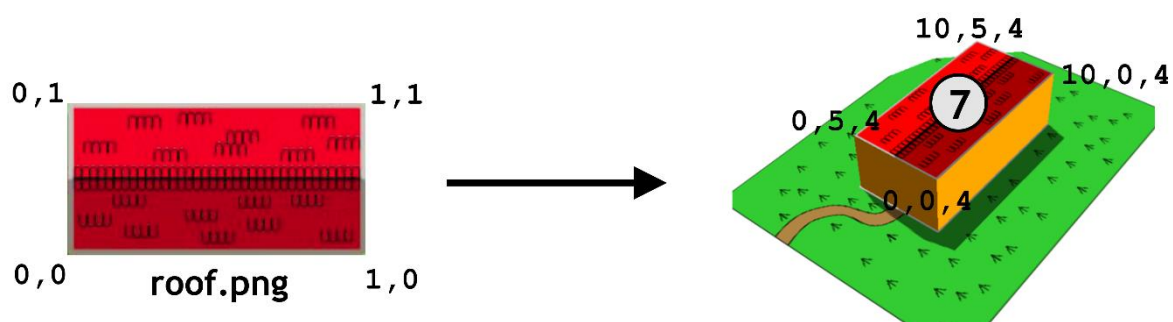


Figure 32: Simple example explaining texture mapping using texture coordinates

TEXTUREPARAM				
SURFACE_GEOMETRY_ID	IS_TEXTURE_PARAMETERIZATION	WORLD_TO_TEXTURE	TEXTURE_COORDINATES	SURFACE_DATA_ID
10	1	NULL	GEOMETRY	20
...

Table 8: Example for table TEXTUREPARAM

Texture coordinates are applicable to polygonal surfaces, whose boundaries are described by a closed linear ring (last coordinate is equal to first). Coordinates are stored with a geometry data type. The `WORLD_TO_TEXTURE` attribute defines a transformation matrix from a location in world space to texture space. For more details see the CityGML Implementation Specification [Gröger et al. 2012].

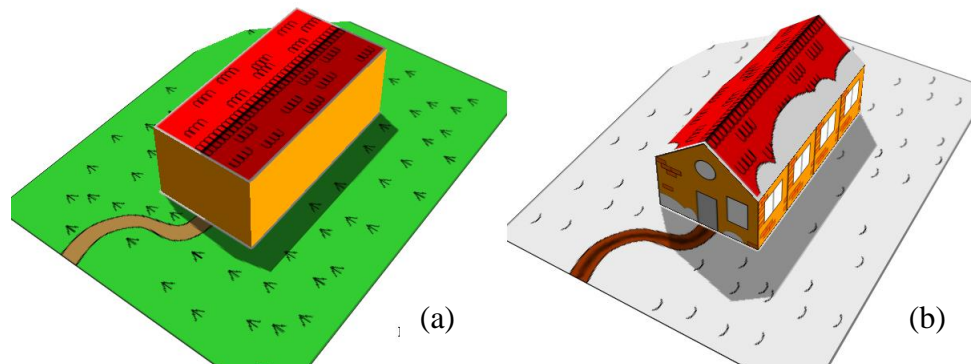


Figure 33: Visualisation of a simple building in LoD1 and LoD2 using the appearance model. Two themes are defined for the building and the surrounding terrain: (a) building in summertime and (b) building in wintertime

Six surface representations are listed in table `SURFACE_DATA` (cf. Table 11). First of all, a homogeneous material is defined (`ID=1`), represented by a 3-component (RGB) colour value which will be used for both appearances (summer and winter). This also applies to a general side façade texture (`ID=3`, Figure 34 right) which is repeated (wrapped) to fill the entire surface. For each of the front side, the back side and the ground two images are available: parameterized ones for the sides (Figure 34 left and middle) and georeferenced ones for the ground and the roof surfaces (Figure 36). The information of textures is stored in a separate table `TEX_IMAGE`. The coordinates for mapping the textures to the object are stored in table `TEXTUREPARAM`. For the general side texture (`SURFACE_DATA_ID=3`) five coordinate pairs are needed to define a closed ring (here: rectangle). Table `SURFACE_GEOMETRY` contains the information of all geometry parts that form the building and its appropriate 3D coordinates (cf. tables on the next page).

See the following page for an example of the storage of appearances in the city database. Figure 34 and Figure 36 show the images used for texturing a building in LoD2. In LoD1, a material definition is used to define the wall colors of the building.

Table 9 to Table 12 show a combination of tables representing the building's textures. There are different images available for summer and winter resulting in two themes: Summer and Winter. The tuples within the tables are color-coded according to their relation to the respective theme:

- Green: only summer related data
- Light-grey: only winter related data
- Orange: both summer and winter related data

Figure 35 shows the LoD2 representation of summer appearances (theme Summer).

APPEARANCE				
ID	GMLID	THEME	CITYMODEL_ID	CITYOBJECT_ID
...
1	App1	Summer		1000
2	App2	Winter		1000
...

Table 9: Excerpt of table APEARANCE

The relation to the building feature is given by the foreign key CITYOBJECT_ID

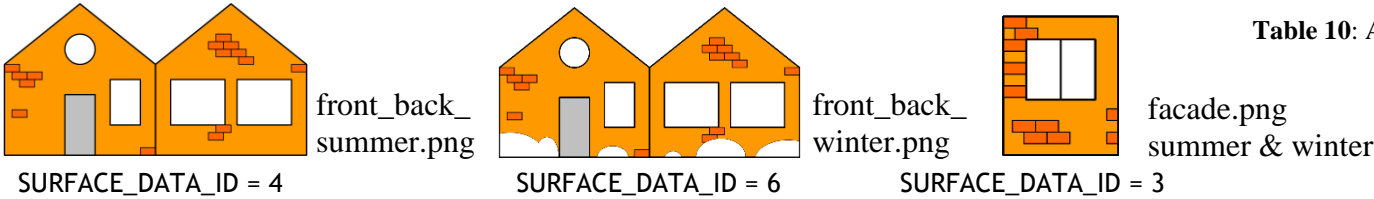


Figure 34: Images for parameterized textures

APPEAR_TO_SURFACE_DATA	
APPEARANCE_ID	SURFACE_DATA_ID
1	7
2	7
1	8
1	3
1	4
2	5
2	3
2	6

Table 10: APPEAR_TO_SURFACE table

COMMENTS
LoD1 S
LoD1 W
LoD2 ground/roof S
LoD2 façade S
LoD2 front/back S
LoD2 ground/roof W
LoD2 façade W
LoD2 front/back W

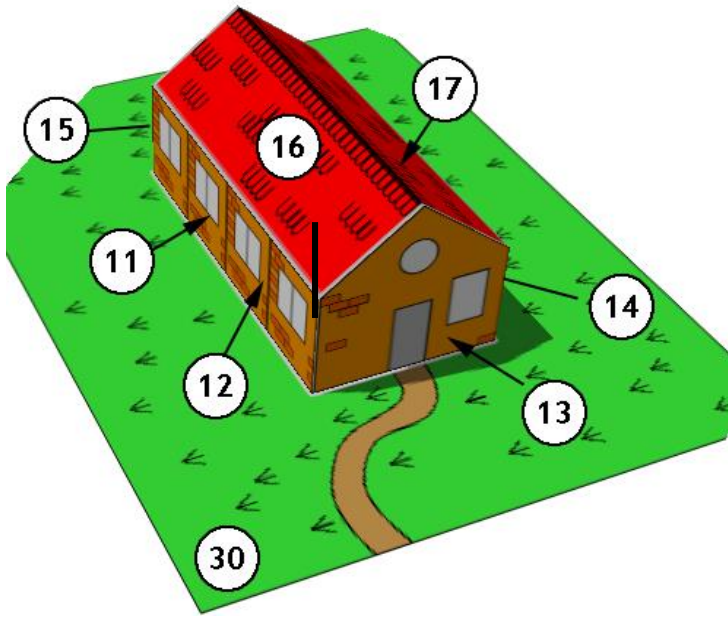


Figure 35: Surface geometries for the building in LoD2 (the IDs for LoD1 are the same as in Figure 29)

SURFACE_DATA							
ID	IS_FRONT	TYPE	X3D_DIFFUSE_COLOR	TEX_IMAGE_ID	TEX_WRAP_MODE	GT_ORIENTATION	GT_REFERENCE_POINT
7	1	X3DMaterial	1.0 0.6 0.0				
3	1	ParameterizedTexture		31	wrap		
4	1	ParameterizedTexture		32	none		
6	1	ParameterizedTexture		33	none		
8	1	GeoreferencedTexture		34	none	0.05 0.0 0.0 0.066667	GEOMETRY
5	1	GeoreferencedTexture		35	none	0.05 0.0 0.0 0.066667	GEOMETRY

Table 11: Excerpt of table SURFACE_DATA and table TEX_IMAGE

TEX_IMAGE		
ID	TEX_IMAGE_DATA	TEX_IMAGE_URI
31	BLOB(...)	facade.png
32	BLOB(...)	front_back_summer.png
33	BLOB(...)	front_back_winter.png
34	BLOB(...)	ground_summer.png
35	BLOB(...)	ground_winter.png

TEXTUREPARAM				
SURFACE_GEOMETRY_ID	IS_TEXTURE_PARA-METRIZATION	WORLD_TO_TEXTURE	TEXTURE_COORDINATES	SURFACE_DATA_ID
30	0	NULL	NULL	8
16	0	NULL	NULL	8
17	0	NULL	NULL	8
13	1	NULL	GEOMETRY	4
15	1	NULL	GEOMETRY	4
12	1	NULL	GEOMETRY	3
11	1	NULL	GEOMETRY	3
14	1	-0.4 0.0 0.0 1.0 0.0 0.0 0.3333 0.0 0.0 0.0 0.0 1.0	NULL	3
30	0	NULL	NULL	5
16	0	NULL	NULL	5
17	0	NULL	NULL	5
13	1	NULL	GEOMETRY	6
15	1	NULL	GEOMETRY	6
2	0	NULL	NULL	7
10	0	NULL	NULL	8

Table 12: Table TEXTUREPARAM

COMMENTS
LoD 2 ground S
LoD 2 roof left S
LoD 2 roof right S
LoD 2 front S
LoD 2 back S
LoD 2 façade left S/W
LoD 2 façade right S/W
LoD2 ground W
LoD 2 roof left W
LoD 2 roof right W
LoD 2 front W
LoD 2 back W
LoD1 walls S/W
LoD1 roof S/W

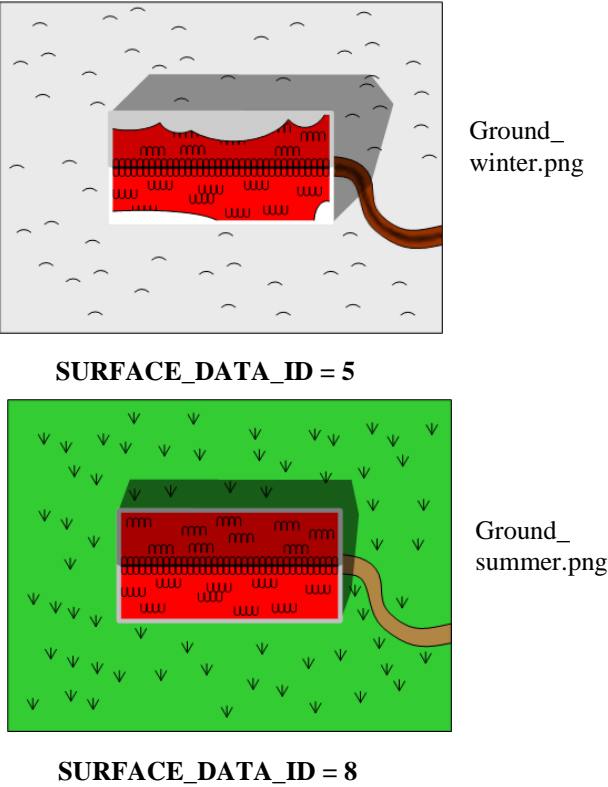


Figure 36: Images for georeferenced textures (The image round_winter.png is assigned to the terrain and the roof surfaces of the building both in LoD1 and LoD2 within the winter theme (a), ground_summer.png within the summer theme (b))

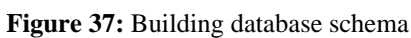


Figure 37: Building database schema

BUILDING

The building model, described in paragraph 2.2.4.2 at the conceptual level, is realised by the tables shown in Figure 37. The three CityGML classes *AbstractBuilding*, *Building* and *BuildingPart* are merged into the single table BUILDING. The subclass relationship with CITYOBJECT arises from using identical IDs, i.e. for each tuple in BUILDING there must exist a tuple within CITYOBJECT with the same ID.

The component hierarchy within a building is realized by the foreign key BUILDING_PARENT_ID which refers to the superordinate building (aggregate) and contains NULL, if such does not exist. This way, a tree-like structure arises also for building aggregates. BUILDING_PARENT_ID points at the predecessor in the tree. The foreign key BUILDING_ROOT_ID refers directly to the top level (root) of a building tree. In order to select all parts forming a building one only has to select those with the same BUILDING_ROOT_ID (cf. table 13).

BUILDING								
ID	BUILDING_PARENT_ID	BUILDING_ROOT_ID	...	LOD0_FOOTPRINT_ID	LOD0_ROOFPRINT_ID	LOD1_MULTI_SURFACE_ID	...	LOD4_SOLID_ID
1	NULL	1		10	NULL	NULL		NULL
2	1	1		NULL	NULL	20		NULL
3	1	1		NULL	NULL	30		NULL
4	2	1		NULL	NULL	NULL		400
5	2	1		NULL	NULL	NULL		500
6	3	1		NULL	NULL	NULL		600
7	3	1		NULL	NULL	NULL		700

Table 13: Tree-like structure for recursive decomposition of buildings

The meaning and the name of most fields are identical to those of the attributes in the UML diagram (cf. Figure 7). Like for CityObjectGroups there are additional _CODESPACE columns for the attributes *class*, *function* and *usage*. A _CODESPACE column is also added for the *roofType* attribute as it is specified as *gml:CodeType* in CityGML. For every attribute including measure information like measuredHeight or storeyHeightsAboveGround etc. an additional _UNIT column is provided to specify the scale measured values are referenced to.

Geometry is represented by several foreign keys LOD0_FOOTPRINT_ID, LOD0_ROOFPRINT_ID, LOD_x_MULTI_SURFACE_ID ($1 \leq x \leq 4$), and LOD_x_SOLID_ID ($1 \leq x \leq 4$) which refer to entries in the SURFACE_GEOMETRY table and represent each LoD's surface geometry.

Optionally the geometry of the terrain intersection curve is stored in the attribute LOD_x_TERRAIN_INTERSECTION ($1 \leq x \leq 4$) using database geometry type (see Table 14). Additional line-typed building elements such as antennas are optionally modelled by the attribute LOD_x_MULTI_CURVE ($1 \leq x \leq 4$, using the same database geometry like for terrain intersection curves).

Geometry storage in Building table - Intersection curves	
Oracle	PostGIS
<ul style="list-style-type: none"> • SDO_GTYPE must have the type <i>MultiCurve</i> / <i>MultiLine</i>, i.e. a composite geometry of different line string segments with 3D coordinates (SDO_GTYPE = 3006) • SDO_ETYPE must be 1 (straight line segments) as curved geometries are not allowed in CityGML and SDO_INTERPRETATION must be 2 	<ul style="list-style-type: none"> • Only MULTILINESTRING Z is allowed, i.e. a composite geometry of different line string segments with 3D coordinates • The geometry type MULTICURVE is not used as CityGML does not allow geometry with arcs

Table 14: Storage of composite line string geometry

THEMATIC_SURFACE

The table THEMATIC_SURFACE represents thematic boundary features. CityGML class *_BoundarySurface* has a number of concrete subclasses representing different types of surfaces. One possibility would be to represent each of these classes by its own table. Here, we choose the approach to create one table representing all those classes. No own tables for the subclasses of *_BoundarySurface* were created in the table schema; instead, the type of the boundary surface is given by the foreign key OBJECTCLASS_ID in the table THEMATIC_SURFACE. Allowed integer values:

- '30' (CeilingSurface)
- '31' (InteriorWallSurface)
- '32' (FloorSurface)
- '33' (RoofSurface)
- '34' (WallSurface)
- '35' (GroundSurface)
- '36' (ClosureSurface)
- '60' (OuterCeilingSurface)
- '61' (OuterFloorSurface)

The aggregation relation between buildings and the corresponding boundary surfaces results from the foreign key BUILDING_ID of the table THEMATIC_SURFACE which refers to the ID of the respective building. The same applies to references between surfaces of building installations (BUILDING_INSTALLATION_ID) and rooms (ROOM_ID). Thematic surfaces and the corresponding parent feature should share their geometry: the geometry should be defined only once and be used conjointly as XLinks. The SURFACE_GEOMETRY, which for example geometrically defines a roof, should at the same time be a part of the volume geometry of the parent feature the roof belongs to.

Example:

In Figure 38 a building geometry is shown which consists of several MultiSurfaces. Please notice, that the left wall (ID 5) is composed of two polygons (IDs 11 and 12) and the roof is splitted into a left and right part (IDs 20 and 21) each of them formed by two parts, the roof

surface and an overhang. In table `SURFACE_GEOMETRY` (cf. Table 15) the value 1 is set for attribute `IS_COMPOSITE` characterising tuples with IDs 5, 20 and 21 as combined surfaces. For adding semantic information table `THEMATIC_SURFACE` is needed, which contains a tuple that references ID 30 in table `SURFACE_GEOMETRY`. The attribute `OBJECTCLASS_ID` with value '33' identifies the related tuple (ID 30) as `RoofSurface` (cf. Table 17). As can be seen in Table 15, the `IS_XLINK` mechanism is used in table `SURFACE_GEOMETRY`. Copies of the left roof (ID 21) and its related parts (IDs 16 and 18) are stored as new tuples (IDs 31 to 33) and are marked by the `IS_XLINK` flag, as well as the original tuple (ID 21).

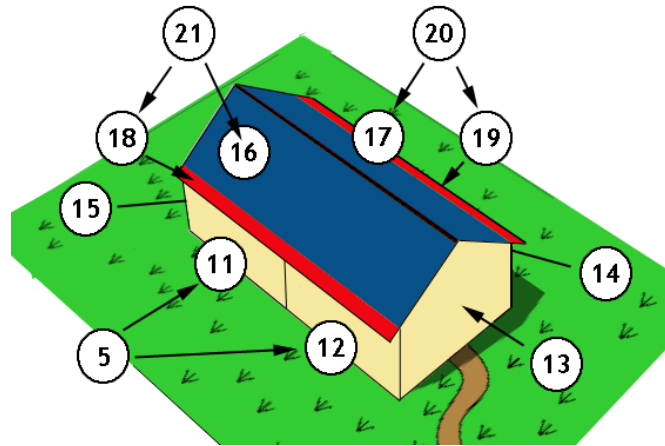


Figure 38: LoD2 building with roof overhangs, highlighted in red

SURFACE_GEOMETRY (excerpt)							
ID	GMLID	PARENT_ID	ROOT_ID	IS_SOLID	IS_COMPOSITE	IS_XLINK	GEOMETRY
3	UUID_LoD2	NULL	3	0	0	0	NULL
5	Left_Wall	3	3	0	1	0	NULL
11	Left_Wall_1	5	3	0	0	0	Geometry comp (5-1) surface 11
12	Left_Wall_2	5	3	0	0	0	Geometry comp (5-2) surface 12
13	Front	3	3	0	0	0	Geometry surface 13
14	Right_Wall	3	3	0	0	0	Geometry surface 14
15	Back	3	3	0	0	0	Geometry surface 15
16	Roof_part_1	21	3	0	0	0	Geometry surface 16
17	Roof_part_2	20	3	0	0	0	Geometry surface 17
18	Overhang_1	21	3	0	0	0	Geometry of overhang 18
19	Overhang_2	20	3	0	0	0	Geometry of overhang 19
20	Roof_right	3	3	0	1	0	NULL
21	Roof_left	3	3	0	1	1	NULL
30	ROOF_SURFACE	NULL	30	0	0	0	NULL
31	Roof_left	30	30	0	1	1	NULL
32	Roof_part_1	31	30	0	0	1	Geometry surface 16
33	Overhang_1	31	30	0	0	1	Geometry of overhang 18
...

Table 15: Excerpt of table `SURFACE_GEOMETRY`

Geometry objects are stored as database geometry datatype

BUILDING					
ID	BUILDING_ROOT_ID	...	LOD1_MULTI_SURFACE_ID	LOD2_MULTI_SURFACE_ID
...		
1	1	1	3
....		

Table 16: Excerpt of table BUILDING

THEMATIC_SURFACE						
ID	OBJECTCLASS_ID	BUILDING_ID	ROOM_ID	LOD2_MULTI_SURFACE_ID	...
...
70	33	1	NULL	30
....

Table 17: Excerpt of table THEMATIC_SURFACE

BUILDING_INSTALLATION

The UML classes *BuildingInstallation* and *IntBuildingInstallation* are realized by the single table **BUILDING_INSTALLATION**. Internal and external objects are distinguished by the attribute **OBJECTCLASS_ID** (external 27, internal 28). The relation to the corresponding parent feature arises from the foreign key **BUILDING_ID** or **ROOM_ID**, whereas the surface based geometry in LoD 2 to 4 is given via the foreign keys **LOD_x_BREP_ID** ($2 \leq x \leq 4$) referring to the table **SURFACE_GEOMETRY**.

Additional point- or line-typed building installation elements such as antennas can be modelled by the attribute **LOD_x_OTHER_GEOM** ($2 \leq x \leq 4$) using the database geometry type (any **GTYPE**, **ETYPE** etc. in Oracle and **GEOMETRY Z** in PostGIS). Since CityGML 2.0.0 building installations can also be represented by using prototypes which are stored as library objects implicitly. The information needed for mapping prototype objects to buildings consists of a base point geometry (**LOD_x_IMPLICIT_REF_POINT** ($2 \leq x \leq 4$)), a transformation matrix (**LOD_x_IMPLICIT_TRANSFORMATION** ($2 \leq x \leq 4$)), which is stored as a string, and a foreign key reference to the **IMPLICIT_GEOMETRY** table (**LOD_x_IMPLICIT_REP_ID** ($2 \leq x \leq 4$)) where a reference to an explicit surface based geometry in LoD 2 to 4 is saved.

OPENING

Openings (CityGML class *Opening*) are represented by the table **OPENING** and are only allowed in LoD3 and 4. No individual tables are created for the subclasses. Instead, the differentiation is achieved by the foreign key **OBJECTCLASS_ID** which refers to the attribute ID of the (meta) table **OBJECTCLASS**. Valid integer values are '39' (Door) and '38' (Window).

Table **OPENING_TO_THEM_SURFACE** associates an opening ID in table **OPENING** with a thematic surface ID in table **THEMATIC_SURFACE** representing the m:n relation between both tables. An address can be assigned to a door (table **OPENING**) by the foreign key **ADDRESS_ID** in the table **OPENING**. Furthermore, addresses may be assigned to buildings (see table **ADDRESS** for detailed information).

Like with building installations openings can be modelled via implicit geometry since CityGML 2.0.0. Thus, the OPENING table does contain the columns LODx_IMPLICIT_REP_ID, LODx_IMPLICIT_REF_POINT and LODx_IMPLICIT_TRANSFORMATION, too.

ROOM

Room objects are allowed in LoD4 only. Therefore the only keys LOD4_MULTI_SURFACE_ID and LOD4_SOLID_ID are referring to the table SURFACE_GEOMETRY. Additionally the foreign keys to tables BUILDING and CITYOBJECT are necessary to map the relationship to these tables.

BUILDING_FURNITURE

As rooms may be equipped with furniture (chairs, wardrobes, etc.), a foreign key referencing to ROOM_ID is mandatory. The geometry of furniture objects can be described explicitly using the attribute LOD4_OTHER_GEOM representing the point- or line-typed entities or using the foreign key LOD4_BREP_ID referring to the table SURFACE_GEOMETRY. Alternatively, the geometry of furniture objects may be represented by using prototypes (ImplicitGeometry) which are stored as library objects. Again, the information needed for mapping prototype objects to rooms consists of a base point, a transformation matrix and a reference to the IMPLICIT_GEOMETRY table.

ADDRESS, ADDRESS_TO_BUILDING, and ADDRESS_SEQ

Addresses are realized by the table ADDRESS. The m:n relation with buildings arises from the table ADDRESS_TO_BUILDING which associates a building ID and an address ID. An address can also be assigned to a door (table OPENING) by the foreign key ADDRESS_ID in the table OPENING. The same applies to addresses of bridges (incl. a table ADDRESS_TO_BRIDGE) and bridge openings.

The next available ID for the table ADDRESS is provided by the sequence ADDRESS_SEQ.

2.3.2.5 Bridge Model

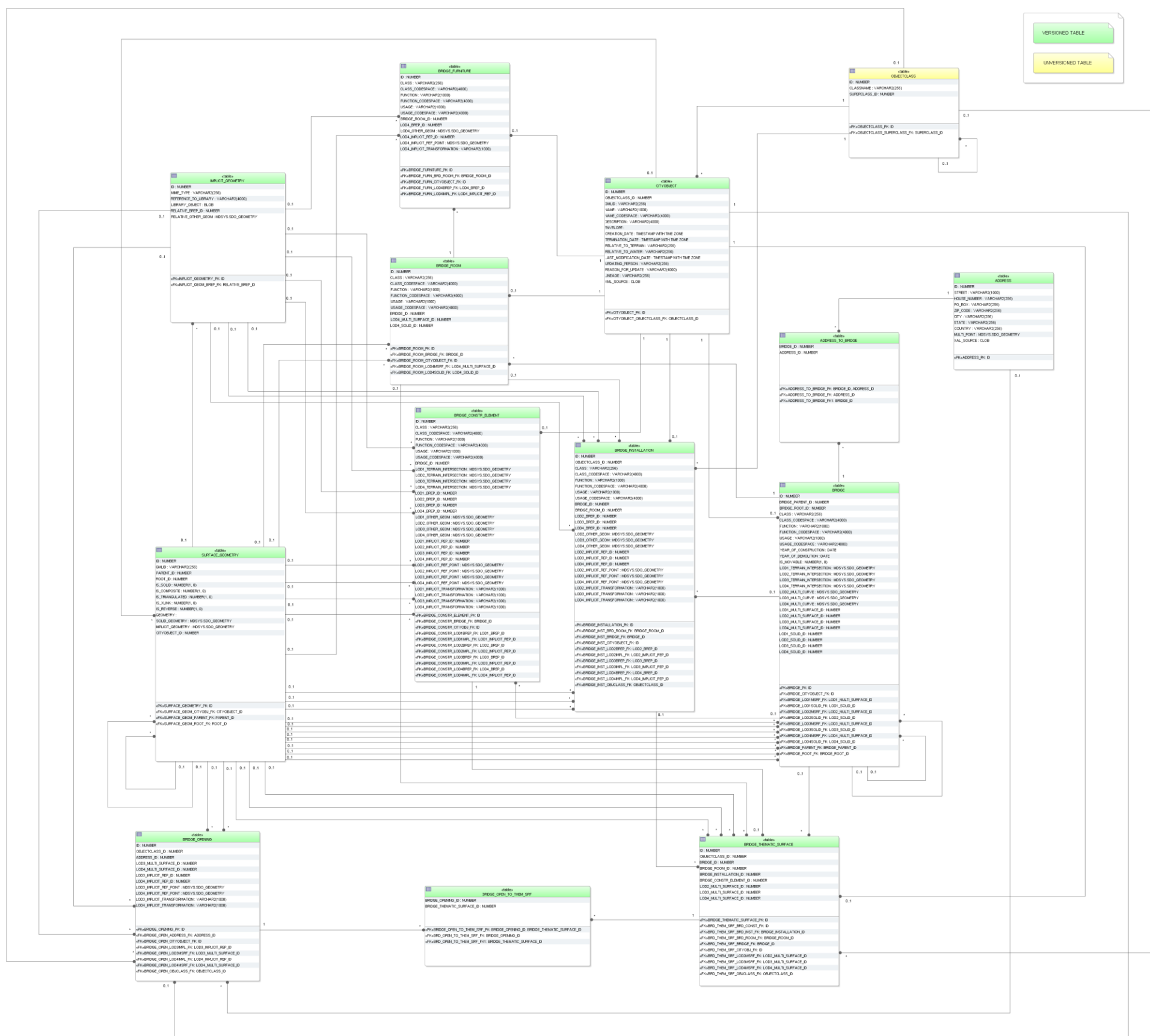


Figure 39: Bridge database schema

The bridge model, described in paragraph 2.2.4.3 at the conceptual level, is realised by the tables shown in Figure 39. The relational schema is identical to the building schema for the most parts except for the naming. Please, refer to the explanation of the building schema on the previous pages for a complete understanding. The main differences to the building schema are the following:

- Bridges cannot be modelled in LoD 0. Therefore, no corresponding columns appear in the BRDGE table.
- CityGML features of bridges, such as boundary surfaces, installations, openings, rooms and furniture, are mapped to separate specific tables and are not stored in already existent ones (e.g. THEMATIC_SURFACE, OPENING, ROOM). The reason for this is to provide a schema that is as close to the UML model as possible. There are slight differences between the building and the bridge model that would lead to ambiguous references e.g. a boundary surface of the building namespace cannot reference to a bridge construction element.
- OBJECTCLASS_ID of table BRIDGE_THEMATIC_SURFACE allows the values:
 - '68' (BridgeCeilingSurface),
 - '69' (InteriorBridgeWallSurface)
 - '70' (BridgeFloorSurface),
 - '71' (BridgeRoofSurface),
 - '72' (BridgeWallSurface),
 - '73' (BridgeGroundSurface),
 - '74' (BridgeClosureSurface),
 - '75' (OuterBridgeCeilingSurface),
 - '76' (OurterBridgeFloorSurface).
- In the BRIDGE_INSTALLATION table external bridge installations can be identified by the OBEJCTCLASS_ID 65 and internal ones by 66.
- The CityGML class *BridgeConstructionElement* is represented by the table BRIDGE_CONSTR_ELEMENT. Its schema is analogue to the BRIDGE_INSTALLATION table for the most parts. The relation to the corresponding bridge results from the foreign key BRIDGE_ID. Explicit and implicit geometry or a decomposition through boundary surfaces is possible. Additionally terrain intersections curves of construction elements can also be stored. Unlike building installations there are no interior construction elements for bridges in CityGML. Therefore, an OBJECTCLASS_ID column is not necessary.
- The OBJECTCLASS_ID column in table BRIDGE_OPENING can be of integer value '79' (BridgeDoor) or '78' (BridgeWindow). They are associated to entries in the BRIDGE_THEMATIC_SURFACE table via the link table BRIDGE_OPEN_TO_THEM_SRF. Like openings of building bridge openings can have addresses assigned to it.

2.3.2.6 CityFurniture Model

The CityGML feature class CityFurniture and its attributes specified in the UML (cf. Figure 13) diagram are directly mapped the CITY_FURNITURE table and its corresponding columns.

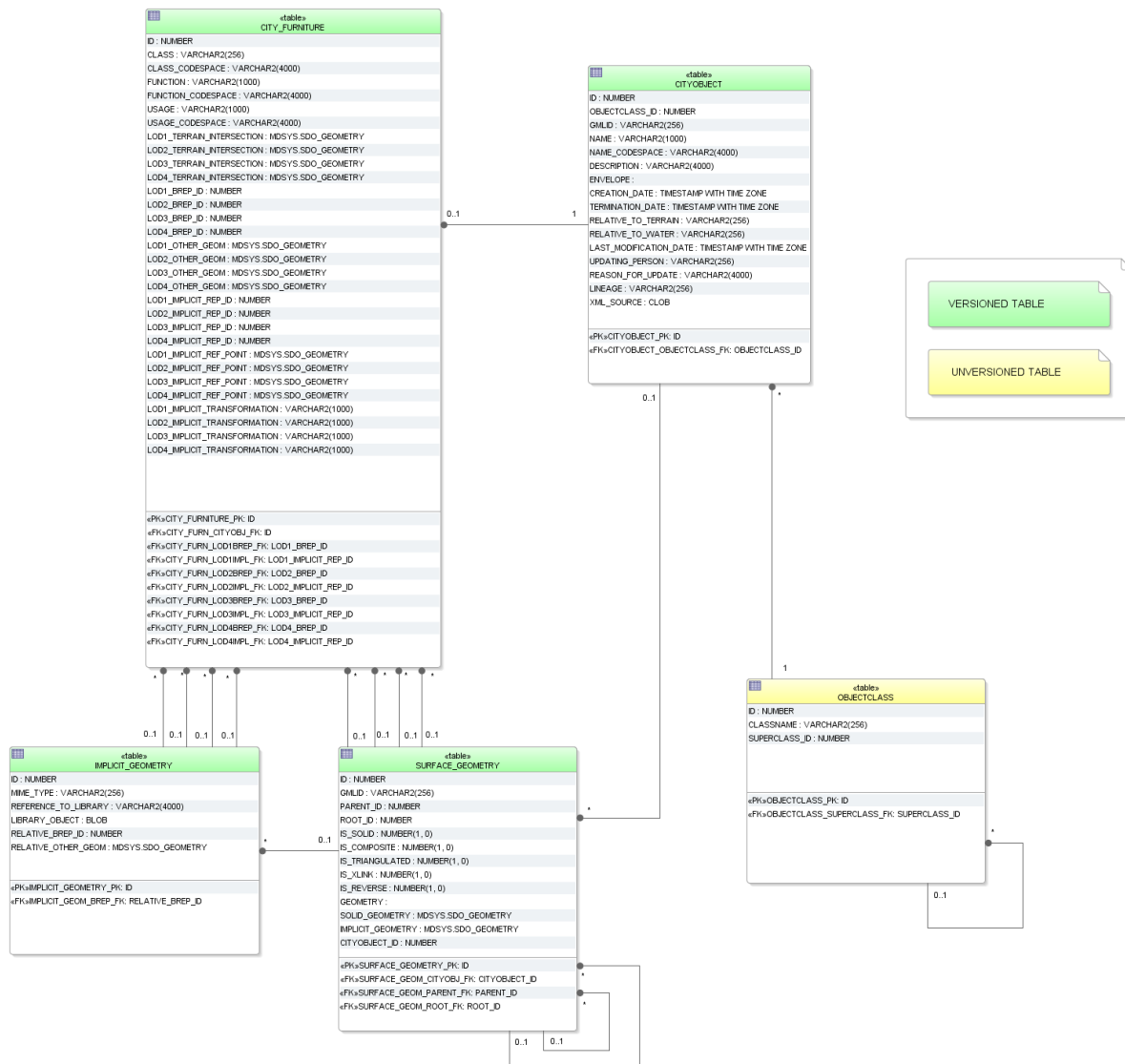


Figure 40: CityFurniture database schema

The geometry of city furniture objects is represented either as a surface-based geometry object (LOD_x_BREP_ID, where $1 \leq x \leq 4$) related to table SURFACE_GEOMETRY, as a point- or line-typed object (LOD_x_OTHER_GEOM, where $1 \leq x \leq 4$) or as implicit geometry LOD_x_IMPLICIT_REF_ID, LOD_x_IMPLICIT_REF_POINT, LOD_x_IMPLICIT_TRANSFORMATION with $1 \leq x \leq 4$). Optionally terrain intersection curves can be stored for city furniture objects.

2.3.2.7 Digital Terrain Model

A tuple in the table RELIEF_FEATURE represents a complex relief object, which consists of different relief components. It has an attribute LOD that describes the affiliation of the relief object to a certain level of detail (LoD) of the city model. The individual components of a complex relief object are stored in the tables BREAKLINE_RELIEF, MASSPOINT_RELIEF, TIN_RELIEF and RASTER_RELIEF. Every relief component has an attribute LOD that describes the affiliation to a certain level of detail (resolution, accuracy). However, individual components of a complex relief object may belong to different LoD and be heterogeneous, i.e. a mixture of TINs, grids and mass points. Optionally, the geometrical separation between the individual relief components of a complex relief object can be realized via polygons (attribute EXTENT), which specify the validity area of the relief component. Every relief component has an attribute NAME that is used for naming of the component. The relief as well as every relief component are derived from CITYOBJECT and receive the same ID as the CityObject. Table RELIEF_FEAT_TO_REL_COMP represents the interrelationship between relief features and relief components.

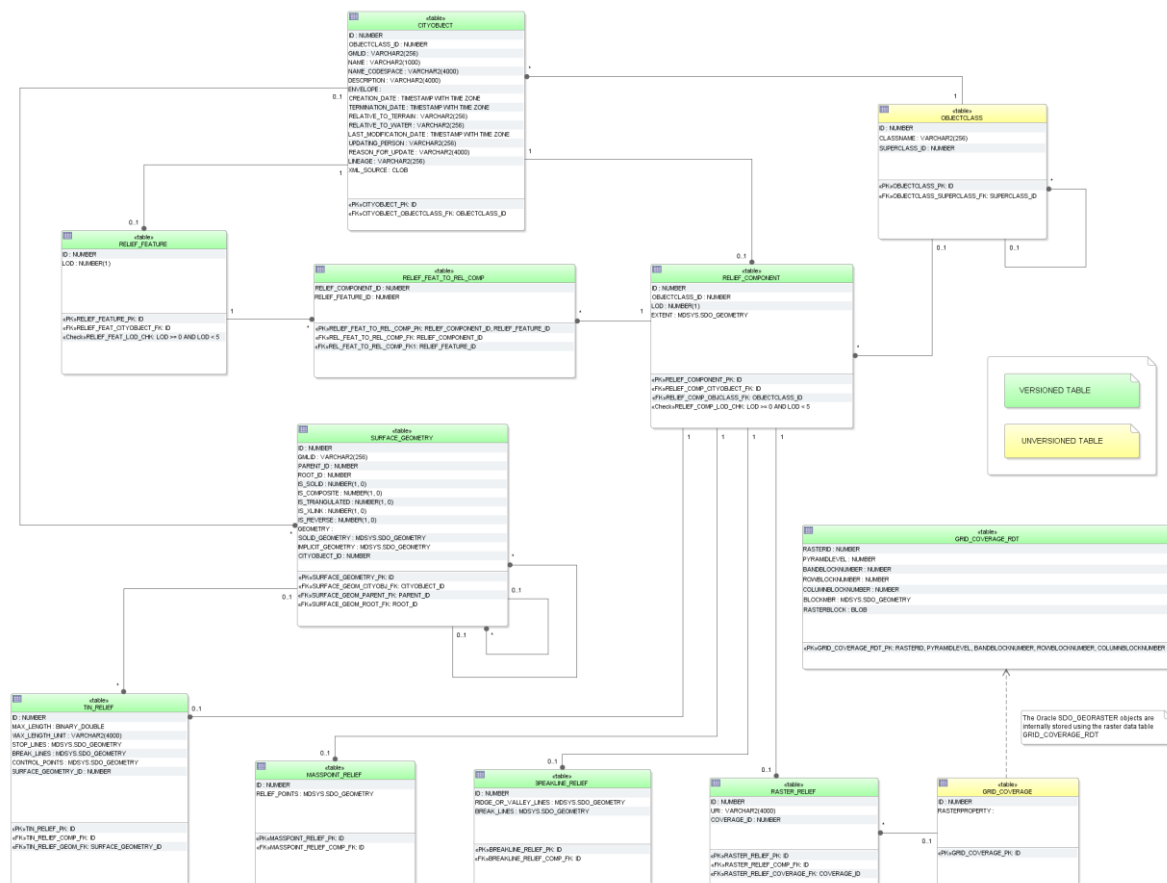


Figure 41: Digital Terrain Model database schema

A raster relief is the only feature in CityGML that can be described by a grid coverage. Corresponding database types are SDO_GEORASTER in Oracle Spatial 11g or higher (not available in Oracle Locator) and RASTER in PostGIS 2.0 or higher. In Oracle for each table that stores SDO_GEORASTER an additional table of type SDO_RASTER is mandatory (raster data table = RDT). It stores the metadata of the SDO_GEORASTER.

In case of that a grid representation is introduced to other features in CityGML in the future, numerous RDT tables would be created when storing grids along with the thematic tables. Thus, a central table called GRID_COVERAGE is used to register all grid data and to prevent numerous additional tables in the 3DCityDB schema. This concept is analogue to the storage of surface-based geometry whereas SURFACE_GEOMETRY is the central table.

Since Oracle Spatial 11g the SDO_GEORASTER type supports Oracle Workspace Manager. Therefore, the table GRD_COVERAGE_RDT can be versioned for history management. However, Oracle Spatial doesn't allow user to version-enable the tables, where GeoRaster objects are stored. Hence, the table GRID_COVERAGE cannot be versioned using the Oracle Workspace Manager.

Geometry attributes for different relief components are limited to the following value domains:

BREAKLINE_RELIEF

- BREAK_LINES and RIDGE_OR_VALLEY_LINES
 - Oracle: MultiLine (GTYPE 3006)
 - PostGIS: MultiLineString Z

TIN_RELIEF

- STOP_LINES and BREAK_LINES
 - Oracle: MultiLine (GTYPE 3006)
 - PostGIS: MultiLineString Z
- RELIEF_POINTS
 - Oracle: MultiPoint (GTYPE 3001 or 3005)
 - PostGIS: MultiPoint Z
- TIN
 - TIN triangles could be stored as triangulated surfaces in table SURFACE_GEOMETRY

MASSPOINT_RELIEF

- RELIEF_POINTS
 - Oracle: MultiPoint (GTYPE 3001 or 3005)
 - PostGIS: MultiPoint Z

RELIEF_COMPONENT

- EXTENT (defines the validity extents of each relief component)
 - Oracle: Polygon (GTYPE 3003, ETYPE 1003, SDO_INTERPRETATION 1 or 3 (optimized rectangle))
 - PostGIS: Polygon Z

2.3.2.8 Generic City Object Model

3D city models will most likely contain attributes, which are not explicitly modelled in CityGML. Moreover, there may be 3D objects that are not covered by the thematic classes of CityGML. Generic objects and attributes help to support the storage of such data.

GENERIC_CITYOBJECT

For generic objects the full variety of different geometrical representations known from other tables is offered. Explicit (LOD_x_BREP_ID, LOD_x_OTHER_GEOM) and implicit geometry (LOD_x_IMPLICIT_REF_ID, LOD_x_IMPLICIT_REF_POINT, LOD_x_IMPLICIT_TRANSFORMATION) as well as terrain intersection curves (LOD_x_TERRAIN_INTERSECTION) (all with $0 \leq x \leq 4$).

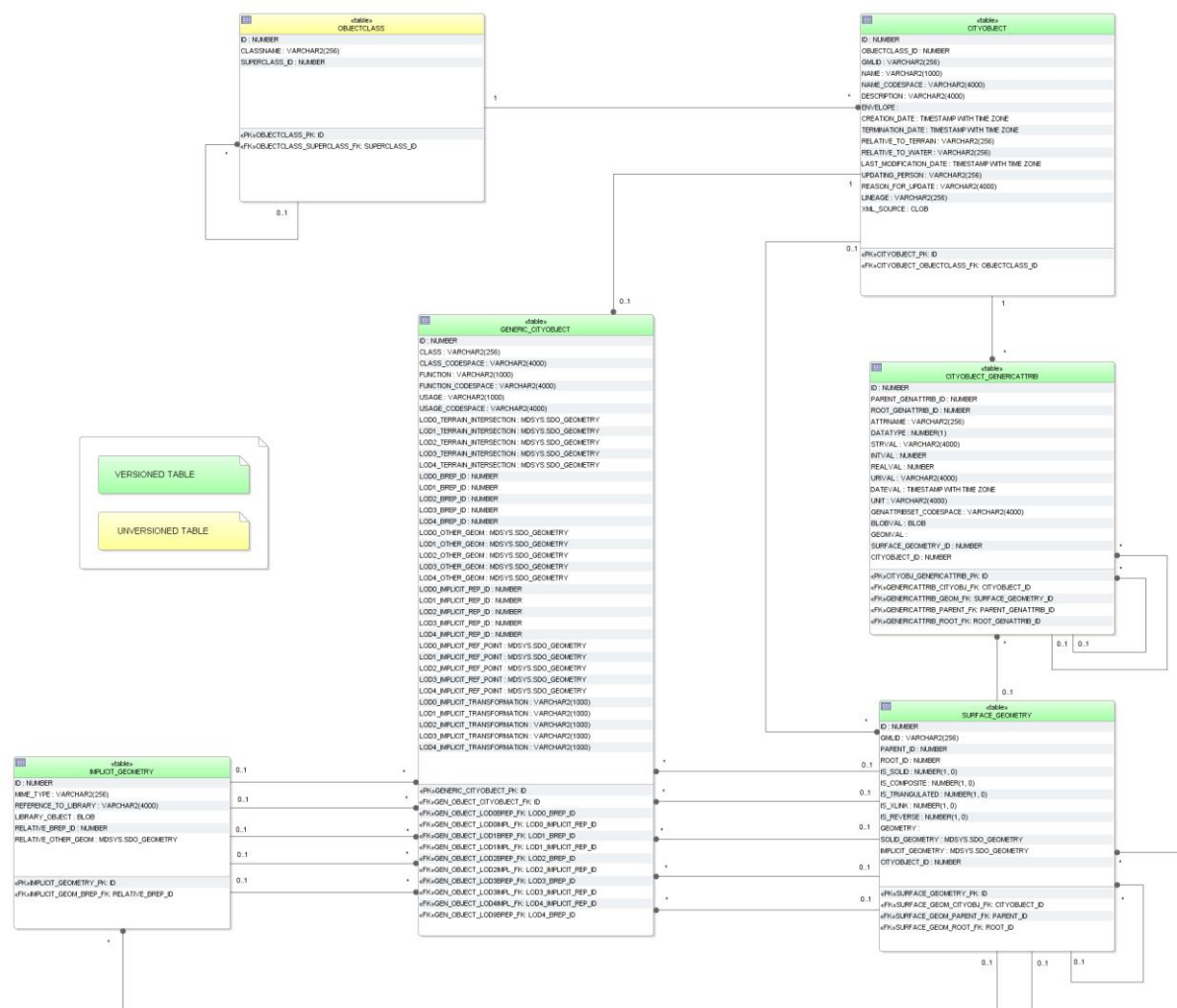


Figure 42: GenericCityObject and generic attributes database schema

CITYOBJECT_GENERICATTRIB, CITYOBJECT_GENERICATT_SEQ

The table CITYOBJECT_GENERICATTRIB is used to represent the concept of generic attributes. However, the creation of a table for every type of attribute was omitted. Instead a single table CITYOBJECT_GENERICATTRIB represents all types and the types are differentiated via the values of the attribute DATATYPE.

The table provides fields for every data type, but only one of those fields is relevant in each case. An overview of the meaning of the entries in the field DATATYPE is given in Table 18. The relation between the generic attribute and the corresponding CityObject is established by the foreign key CITYOBJECT_ID.

DATATYPE	attribute type
1	STRING
2	INTEGER
3	REAL
4	URI
5	DATE
6	UNIT
7	Group of generic attributes
8	BLOB
9	Geometry type
10	Geometry via surfaces in the table SURFACE_GEOMETRY

Table 18: Attribute type

Please note that the binary and geometric data types (incl. geometry via surfaces) are not supported by CityGML and cannot be exported using the CityGML Import / Export tool! But, if a user wants to add additional attributes to thematic tables, he should use the schema of the CITYOBJECT_GENERICATTRIB table rather than adding additional columns to existing tables, because only in this way the Import / Export tool can automatically write them to CityGML.

Moreover, generic attributes can be grouped using the CityGML class *genericAttributeSet*. Since *genericAttributeSet* itself is a generic attribute, it may also be contained in a generic attribute set facilitating a recursive nesting of arbitrary depth. This hierarchy within a *genericAttributeSet* is realized by the foreign key PARENT_GENATTRIB_ID which refers to the superordinate *genericAttributeSet* (aggregate) and contains NULL, if such does not exist. The foreign key ROOT_GENATTRIB_ID refers directly to the top level (root) of a *genericAttributeSet* tree. In order to select all generic attributes forming a *genericAttributeSet* one only has to select those with the same ROOT_GENATTRIB_ID.

The next available ID for the table CITYOBJECT_GENERICATTRIB is provided by the sequence CITYOBJECT_GENERICATT_SEQ.

2.3.2.9 LandUse Model

The CityGML feature class LandUse and its attributes specified in the UML (cf. Figure 17) diagram are directly mapped the LAND_USE table and its corresponding columns. The relation to table SURFACE_GEOMETRY is established by the foreign keys LODx_MULTI_SURFACE_ID, where $0 \leq x \leq 4$.



Figure 43: LandUse database schema

2.3.2.10 Transportation Model

For the realisation of transportation objects two tables are provided: TRAFFIC_AREA and TRANSPORTATION_COMPLEX.

TRAFFIC_AREA

Next to the common attribute triple *class*, *function* and *usage* traffic areas can store information about their *surfaceMaterial*. In the UML model this attribute is specified as *gml:CodeType* which makes an additional *_CODESPACE* column necessary. The representation of geometry is handled by foreign keys LODx_MULTI_SURFACE_ID (with $2 \leq x \leq 4$). The aggregation relation between a transportation complex and the corresponding traffic areas results from the foreign key TRANSPORTATION_COMPLEX_ID. The foreign key OBJECTCLASS_ID indicates whether a tuple represents a *TrafficArea* (value 47) or an *AuxiliaryTrafficArea* (value 48) feature.

TRANSPORTATION_COMPLEX

As shown in the UML diagram, every traffic area object may have the attributes *class*, *function* and *usage*. For differentiation between the subclasses an OBJECTCLASS_ID column is used again:

- ‘42’ *TransportationComplex*
- ‘43’ *Track*
- ‘44’ *Railway*
- ‘45’ *Road*
- ‘46’ *Square*

In the coarsest level transportation complexes are modelled by line objects. The corresponding column is called LOD0_NETWORK of geometry type MultiCurve in Oracle and MultiLineString Z in PostGIS. Starting from LOD1 the representation of object geometry is handled by foreign keys LODx_MULTI_SURFACE_ID (with $1 \leq x \leq 4$).

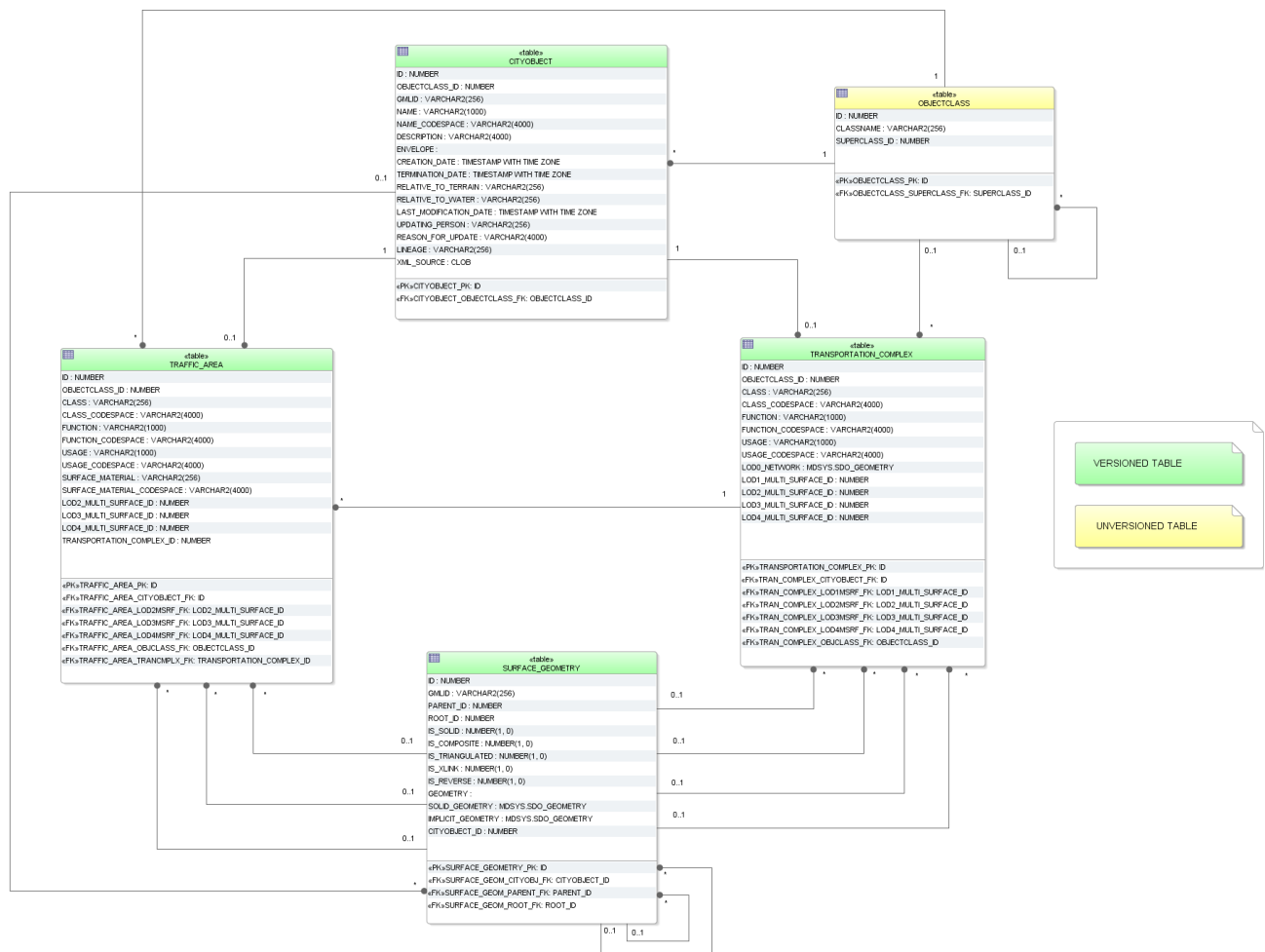


Figure 44: Transportation database schema

2.3.2.11 Tunnel Model

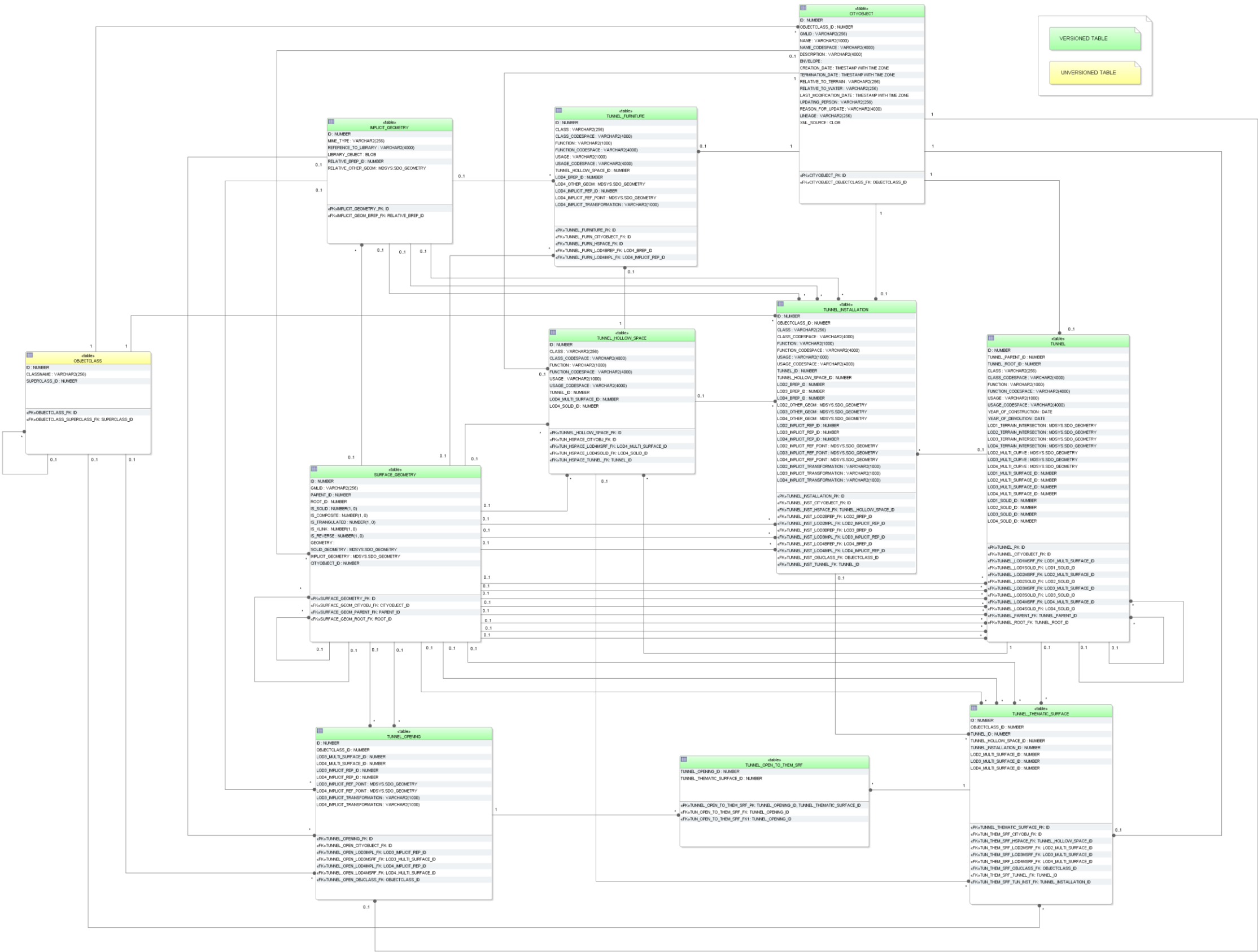


Figure 45: Tunnel database schema

The tunnel model, described in paragraph 2.2.4.9 at the conceptual level, is realised by the tables shown in Figure 45. The relational schema is identical to the building and bridge schema for the most parts except for the naming. Please, refer to the explanation of the building schema on the previous pages for a complete understanding. The main differences to the building schema are the following:

- Tunnels cannot be modelled in LoD 0. Therefore, no corresponding columns appear in the TUNNEL table.
- The CityGML feature HollowSpace can be seen analogue to the feature room of a building or a bridge
- CityGML features of tunnels, such as boundary surfaces, installations, openings, hollow spaces and furniture, are mapped to separate specific tables and are not stored in already existent ones (e.g. THEMATIC_SURFACE, OPENING). The reason for this is to provide a schema that is as close to the UML model as possible. There are slight differences between the building and the tunnel model that would lead to ambiguous references e.g. a boundary surface of the building namespace cannot reference to a tunnel feature.
- OBJECTCLASS_ID of table TUNNEL_THEMATIC_SURFACE allows the values:
 - '89' (TunnelCeilingSurface),
 - '90' (InteriorTunnelWallSurface)
 - '91' (TunnelFloorSurface),
 - '92' (TunnelRoofSurface),
 - '93' (TunnelWallSurface),
 - '94' (TunnelGroundSurface),
 - '95' (TunnelClosureSurface),
 - '96' (OuterTunnelCeilingSurface),
 - '97' (OuterTunnelFloorSurface).
- In the TUNNEL_INSTALLATION table external tunnel installations can be identified by the OBJECTCLASS_ID 86 and internal ones by 87.
- The OBJECTCLASS_ID column in table BRIDGE_OPENING can be of integer value '100' (BridgeDoor) or '99' (BridgeWindow). They are associated to entries in the TUNNEL_THEMATIC_SURFACE table via the link table TUNNEL_OPEN_TO_THEM_SRF. Like openings of building bridge openings can have addresses assigned to it.
- Tunnels and tunnel openings do not have addresses.

2.3.2.12 Vegetation Model

The vegetation model specified in paragraph 2.2.4.10 is realized by the tables shown in Figure 46 which correspond largely to the UML model.

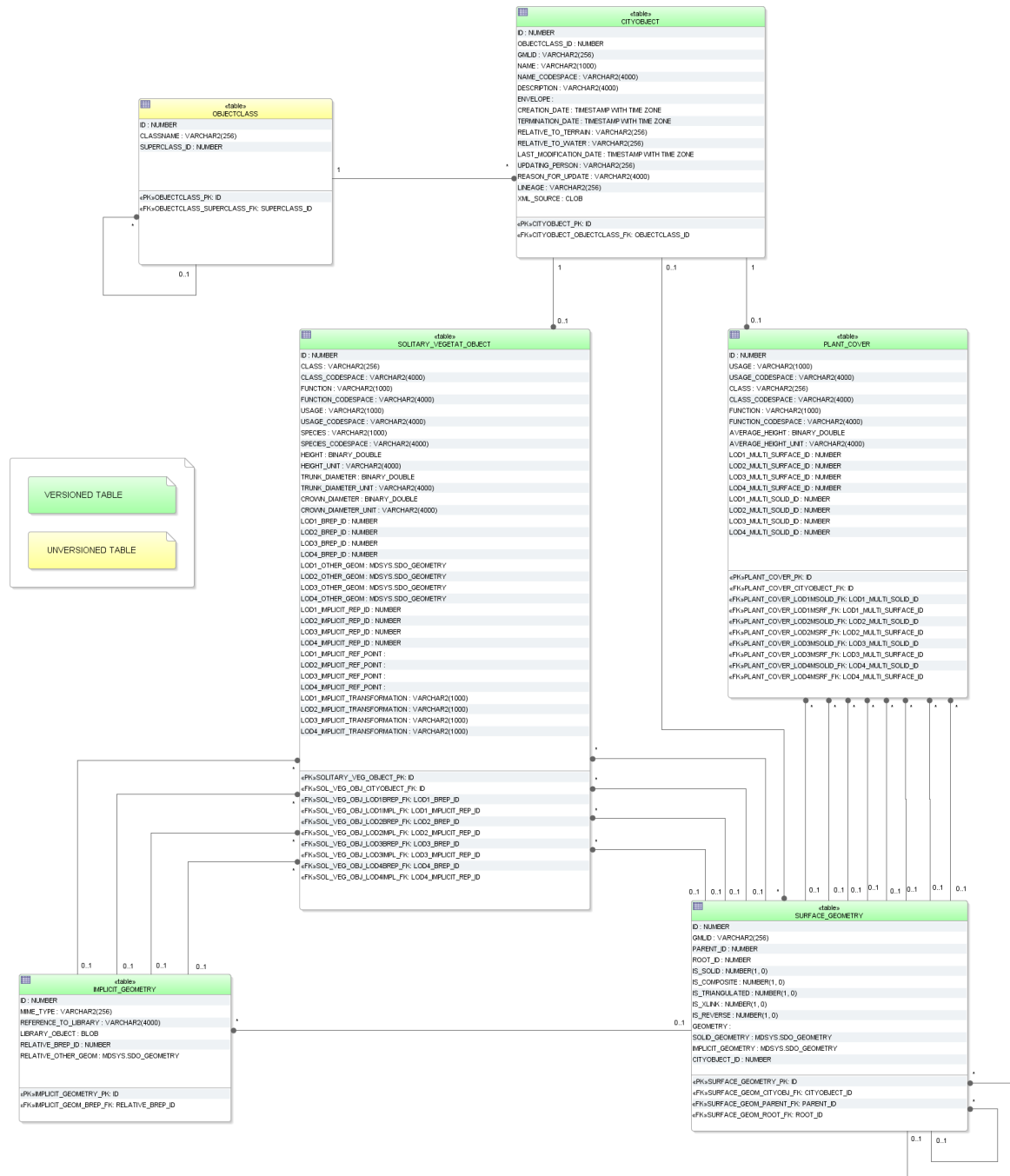


Figure 46: Vegetation database schema

SOLITARY_VEGETAT_OBJECT

The attributes *class*, *function*, *usage*, *species*, *height*, *trunkDiameter*, and *crownDiameter* describe single vegetation objects. The attribute *species* is of type *gml:CodeList* in CityGML that can be referenced to a certain codespace. Therefore another *_CODESPACE* column is provided in the *SOLITARY_VEGETAT_OBJECT* table. Similar to the building table attribute

with measure information can optionally be coupled with a reference to the used measuring scale by an additional `_UNIT` column.

The geometry of the vegetation can either be described explicitly using the attribute `LOD4_OTHER_GEOM` or `LOD4_BREP_ID` or implicitly using a foreign key relation the `IMPLICIT_GEOMETRY` table including a reference point and optionally a transformation matrix (`LODx_IMPLICIT_REP_ID`, `LODx_IMPLICIT_REF_POINT` `LODx_IMPLICIT_TRANSFORMATION`, with $1 \leq x \leq 4$).

PLANT_COVER

Information on vegetation areas are contained in attributes *usage*, *class*, *function*, and *averageHeight*. There is also a `_UNIT` column to specify the scale the *averageHeight* values are based on. The geometry is restricted to a MultiSurface or (and this is unique for PlantCover features) a MultiSolid, represented respectively by the foreign keys `LODx_MULTI_SURFACE_ID` (with $1 \leq x \leq 4$) and `LODx_MULTI_SOLID_ID` which refer to the `SURFACE_GEOMETRY` table.

2.3.2.13 WaterBody Model

WATERBODY, WATERBOD_TO_WATERBND_SRF

The modelling of the WATERBODY database schema corresponds largely to the respective UML model. For LoD0 and LoD1 additional attributes are added, e.g. for modelling river geometry (`LODx_MULTI_CURVE`).

The geometries of LOD0 and LOD1 areal water bodies are stored within the table `SURFACE_GEOMETRY`. The foreign keys `LODx_MULTI_SURFACE_ID` (with $0 \leq x \leq 1$) refer to the corresponding rows. Geometry for water filled volumes is handled in a similar way using foreign keys `LODx_SOLID_ID` (with $1 \leq x \leq 4$).

For mapping the *boundedBy* aggregation which identifies the water body's exterior shell managed by the `WATERBOUNDARY_SURFACE` table, the additional table `WATERBOD_TO_WATERBND_SRF` is needed to realise the m:n relationship.

WATERBOUNDARY_SURFACE

The exterior shell of a *WaterBody* can be differentiated semantically using features of the type *_WaterBoundarySurface*. These features are stored in the `WATERBOUNDARY_SURFACE` table and can be distinguished by the `OBJECTCLASS_ID` attribute:

- '11' (*WaterSurface*)
- '12' (*WaterGroundSurface*)
- '13' (*WaterClosureSurface*)

Since every *_WaterBoundarySurface* object must have at least one associated surface geometry, the foreign keys `LODx_SURFACE_ID` (with $2 \leq x \leq 4$, no MultiSurface here) are used to realise these relations.

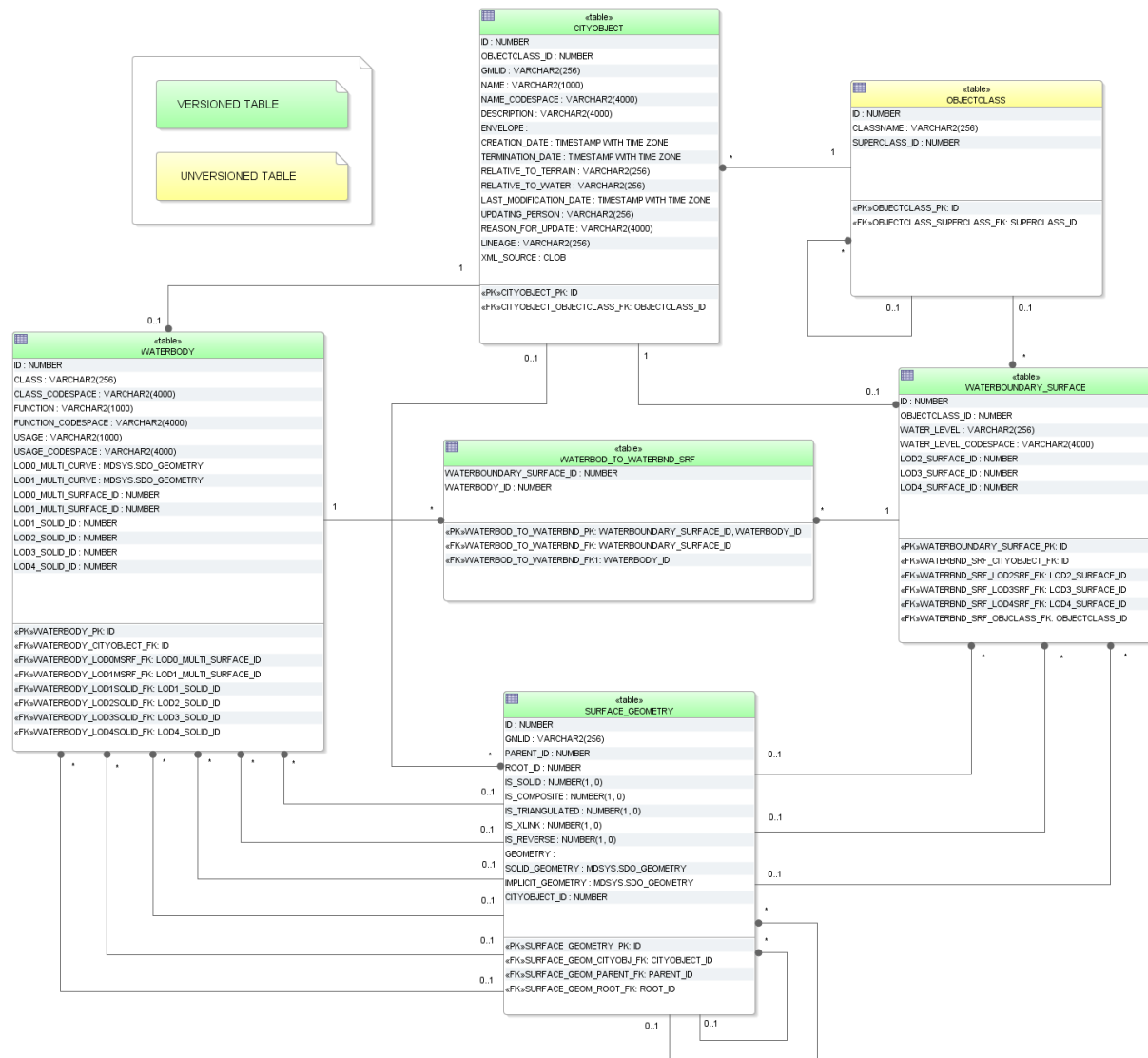


Figure 47: WaterBody database schema

2.3.3 Sequences

Figure 48 lists predefined sequences from which multiple users may generate unique integers for primary keys automatically. Sequences help to coordinate primary keys across multiple rows and tables. For instance, the ID values of the BUILDING table are generated from the CITYOBJECT_SEQ sequence. The sequences are defined to start with 1 and to be incremented by 1 when a sequence number is generated. It is highly recommended to generate ID values for all tables by using the predefined sequences only.

The sequence GRID_COVERAGE_RDT_SEQ does not exist in the PostgreSQL version as the corresponding table does not exist.

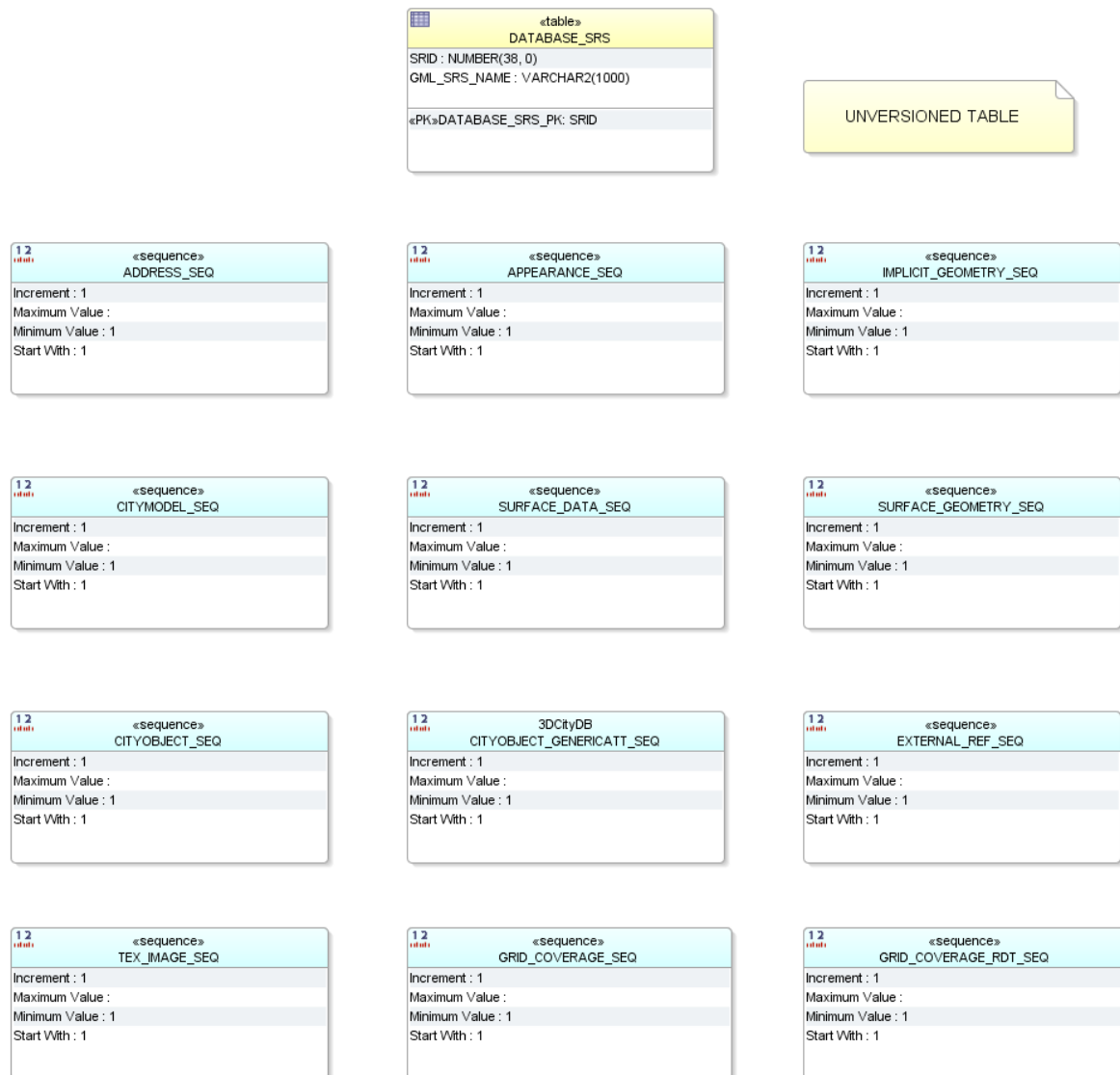


Figure 48: Tables for spatial information and sequences

2.3.4 Definition of the CRS for a 3D City Database instance

The definition of the CRS of a 3D City Database instance consists of two components: 1) a valid *Spatial Reference Identifier* (SRID) (EPSG code) and 2) an OGC GML conformant *definition identifier for the CRS*. Both components are defined during the database setup (see chapter 3.3) and are further stored in the table `DATABASE_SRS` (see Figure 48).

The SRID is an integer value key pointing to spatial reference information within Oracle's `MDSYS.CS_SRS` table or PostGIS' `SPATIAL_REF_SYS` table. Both DBMSs are shipped with a large number of predefined spatial reference systems. **At setup time, the SRID chosen as default value for the 3D City Database instance must already exist in the mentioned tables!**

The GML conformant *CRS definition identifier* is used as value for the `gml:srsName` attribute on GML geometry elements when exporting database contents to CityGML instance documents. It should follow the OGC recommendation for the Universal Resource Name (URN) encoding of CRSs given in the OGC Best Practice Paper *Definition identifier URNs in*

OGC namespace [Whiteside 2009]. At setup time, please make sure to provide a URN value which corresponds to the spatial reference system identified by the default SRID of the database instance. Since CityGML is a 3D standard, the URN encoding **shall always represent a three-dimensional CRS** which, for example, can be denoted as compound coordinate reference systems [Whiteside 2009]. The general syntax of a URN encoding for a compound reference system is as follows:

```
urn:ogc:def:crs,crs:authority:version:code,crs:authority:
version:code
```

Authority, version, and code depend on the information authority providing the CRS definition (e.g. EPSG or OGC). The following example shows a possible combination of an SRID (here referring to a 2D CRS) and CRS URN encoding (3D) to set up an instance of the 3D City Database:

SRID: 31466

URN: urn:ogc:def:crs,crs:EPSG:7.7:31466,crs:EPSG:7.7:5783

The example SRID is referencing a Projected CRS defined by EPSG (DHDN / 3-degree Gauss-Krüger zone 2; used in the western part of Germany; EPSG-Code: 31466). The URN encodes a compound coordinate reference system which adds a Vertical CRS as height reference (DHHN92 height, EPSG-Code: 5783).

3 Implementation and Installation

The 3D City Database comes with SQL scripts for setting up an instance of the relational schema on a spatial database system (Oracle Spatial/Locator or PostgreSQL/PostGIS) and with a database loading and extracting tool called Importer/Exporter. Installers are available for download at <http://www.3dcitydb.org>. The source code of the 3D City Database project is hosted on <https://github.com/3dcitydb>. Please follow the instructions on the next pages in order to complete a proper installation.

3.1 System requirements

3.1.1 3D City Database

Setting up an instance of the 3D City Database requires a running installation of an Oracle or PostgreSQL database server.

Oracle

Supported version are **Oracle 10g R2 or higher**. The 3D City Database requires spatial data support provided either through the Oracle *Spatial* or *Locator* extension. It is highly recommended to install available patches in order to avoid unexpected errors and to benefit from the latest functionality. For Oracle 10g R2, at least patch set 10.2.0.4.0 is required for using the KML/COLLADA export capabilities.

PostgreSQL

Supported versions are **PostgreSQL 9.1 or higher** with the **PostGIS extension 2.0 or higher**. Please also make sure to always install the latest patches and updates.

3.1.2 Importer/Exporter Tool

The Importer/Exporter tool can run on any platform providing support for Java 7 (or higher). It has been successfully tested on (but is not limited to) the following operating systems:

- Microsoft Windows XP, Vista, 7, 8, 8.1;
- Apple Mac OS X 10.6, 10.10.1;
- Ubuntu Linux 9, 10, 11, 14.

Prior to the setup of the Importer/Exporter tool, the **Java 7 Runtime Environment** (or higher) **must be installed on your system**. The necessary installation package can be obtained from <http://www.java.com/en/download>.

The Importer/Exporter is shipped with a universal installer that will guide you through the steps of the setup process. A full installation of the Importer/Exporter including documentation and example CityGML files requires approx. 110 MB of hard disk space. Installing only the mandatory application files will use approx. 22 MB of hard disk space. Installation packages can be selected during the setup process.

The Importer/Exporter requires at least 128 MB of main memory and, per default, is assigned a maximum of 1 GB of main memory. Although these memory limits are chosen to be reasonable on most platforms and for most import/export procedures, especially dealing with large CityGML respectively KML/COLLADA files might require higher memory limits. Note that adapting the available main memory has to be done *manually*. Please refer to chapter 5.1 for more details.

3.2 Installation of the Importer/Exporter and the 3D City Database SQL Scripts

Download the universal installer from <http://www.3dcitydb.org> and save it to your local file system. The universal installer is implemented as executable Java Archive (JAR) file. In order to run the installation wizard, simply double-click on the *3DCityDB-Importer-Exporter-3.0-Setup.jar* file.

After accepting the license agreement and specifying a destination folder, the user can choose the installation packages to be installed besides the mandatory core application files of the Importer/Exporter. It is recommended to at least select the packages ‘3D City Database’ and ‘Documentation’. The ‘3D City Database’ package contains all SQL scripts that are required for setting up an instance of the 3D City Database on your spatial database system. Please refer to chapter 3.3 for a step-by-step guide on how to use these SQL scripts. The package ‘Sample CityGML and KML/COLLADA datasets’ contains license-free sample data that may be used in first tests.

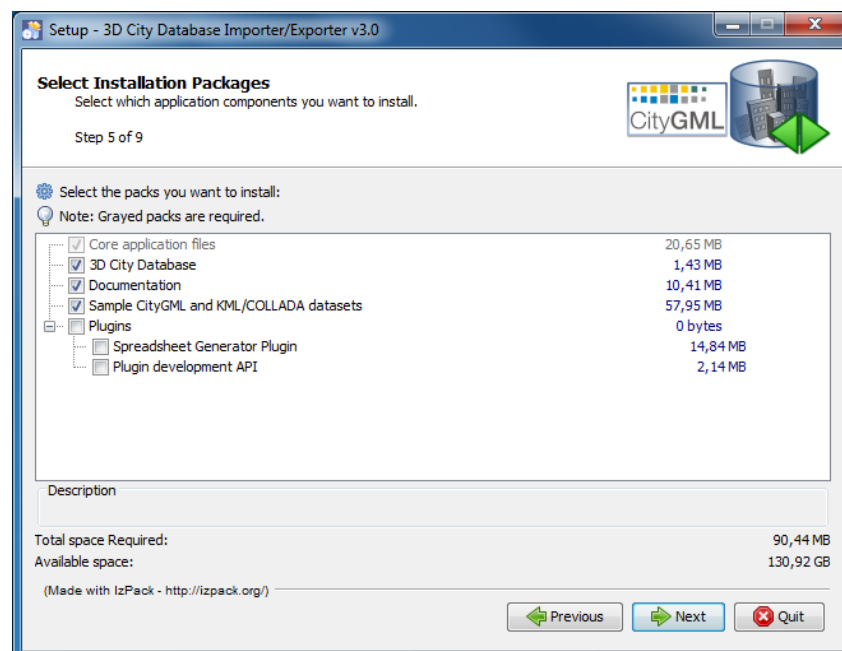


Figure 49: Installation wizard of Import/Export tool (Step 5)

The option ‘*Plugins*’ allows a user to install additional plugins for the Importer/Exporter. With this release, a ‘*Spreadsheet Generator Plugin*’ is available for installation (for documentation see chapter 6.2). More plugins may be added in future releases. Furthermore, a ‘*Plugin*

development API can be copied to your local file system during installation, which enables the Java-based development of plugins.

After successful installation, the contents of all selected installation packages are available in the chosen destination folder. In order to run the Importer/Exporter, simply use the starter script in that same folder (refer to chapter 5.1 for more information).

Note: Before the Importer/Exporter can connect to an Oracle/PostgreSQL database, **the 3D City Database schema has to be set up** on top of your spatial RDBMS, which is explained in the next chapter.

The installation folder should contain the following subfolders and files:

Folder/File	Optional	Explanation
3dcitydb	x	Contains all SQL scripts and stored procedures for operating the 3DCityDB
lib		Contains all libraries required by the Importer/Exporter
licence		Contains the license documents for the 3DCityDB
manual	x	Contains the documentation for the 3DCityDB
plugin-api	x	Contains an API to be used for plugin development for the Importer/Exporter
plugins		Contains plugins of the Importer/Exporter. Plugins only have to be copied to this directory in order to make them available in the application.
samples	x	Contains CityGML and KML/COLLADA test datasets
templates		Contains HTML templates for information balloons for KML/COLLADA exports and a selection of coordinate reference systems in the form of XML documents
uninstaller		Contains a JAR executable that uninstalls the Importer/Exporter
3DCityDB-Importer-Exporter.bat		Starts the application on double-click (Note that a different starter script is installed on UNIX/Linux and MacOS)
impexp.ico		The icon of the Importer/Exporter (may differ on different operating systems)
README.txt		A brief information about the application

Table 19: Content of the installation folder.

3.3 Setting up the database schema

The required scripts for setting up the 3D City Database are located in the installation directory of the Importer/Exporter within the *3dcitydb/oracle/* or *3dcitydb/postgis/* subfolders. The following table provides an overview of their contents:

Folder/File	Oracle	PgSQL	Explanation
CREATE_CITYDB_PKG.sql	x	x	Creates a separate schema 'citydb_pkg' (PostgreSQL). Calls the scripts in the folder CITYDB_PKG
CREATE_DB.bat		x	Batch file that calls CREATE_DB.sql (for MS Windows family)
CREATE_DB.sh		x	Shell script that calls CREATE_DB.sql (UNIX/ Linux & derivations, MacOS X)
CREATE_DB.sql	x	x	Runs all scripts for creating the relational schema of the 3DCityDB (PostgreSQL) or calls CREATE_DB2.sql (Oracle)
CREATE_DB2.sql	x		Runs all SQL scripts for creating the relational schema of the 3DCityDB
DISABLE_VERSIONING.sql	x		Deactivates versioning for a database (execution can be time-consuming); calls DISABLE_VERSIONING2.sql after checking for Spatial or Locator option
DISABLE_VERSIONING2.sql	x		Called by DISABLE_VERSIONING.sql
DROP_CITYDB_PKG.sql	x		Deletes all elements of the GEODB package (in the PostgreSQL version this is done in DROP_DB.sql)
DROP_DB.bat		x	Batch file that calls DROP_DB.sql (for MS Windows family)
DROP_DB.sh		x	Shell script that calls DROP_DB.sql (UNIX/ Linux & derivations, MacOS X)
DROP_DB.sql	x	x	Deletes all elements of the 3DCityDB relational schema (PostgreSQL) or calls DROP_DB2.sql (Oracle)
DROP_DB2.sql	x		Called by DROP_DB.sql
ENABLE_VERSIONING.sql	x		Activates versioning for a database (execution can be time-consuming); calls ENABLE_VERSIONING2.sql after checking for Spatial or Locator option
ENABLE_VERSIONING2.sql	x		Called by ENABLE_VERSIONING.sql
MIGRATION/..	x	x	Contains scripts to migrate content of previous 3DCityDB v2.x to v3.0.0 (see chapter 3.4)
[PL/SQL] [PL/pgSQL]/CITYDB_PKG/..	x	x	Contains scripts that create procedures for functions of the Importer/Exporter (see chapter 4)
SCHEMA/CONSTRAINTS/ CONSTRAINTS.sql	x		Creates foreign key conditions for the table relations
SCHEMA/INDEXES/INDEXES. sql	x		Creates indexes for foreign key and geometry columns
SCHEMA/SEQUENCES/ SEQUENCES.sql	x		Creates sequences for the ID columns of the tables
SCHEMA/TABLES/TABLES.sql	x		Creates all tables of the 3DCityDB
SCHEMA/SCHEMA.sql		x	Creates all sequences, tables, constraints and indexes of the 3DCityDB
UTIL/CREATE_DB/ DO_NOTHING.sql	x		Dummy function
UTIL/CREATE_DB/ HINT_ON_MISSING_SRS.sql	x		Returns corresponding messages when the selected SRID was not found when setting up the database in Oracle Spatial
UTIL/CREATE_DB/ HINT_ON_MISTYPED_DBVER SION.sql	x		Returns corresponding messages when selected Oracle version has neither been 'S' for Spatial or 'L' for Locator
UTIL/CREATE_DB/	x	x	Fills the table 'Objectclass'.

OBJECTCLASS_INSTANCES.sql			
UTIL/SRS/SOLDNER_BERLIN_SRS.sql	x		Script that creates the Berlin reference system on the database server. Already exists in PostGIS.
UTIL/VACUUM/VACUUM_SPATIAL_COLUMNS.sql		x	Script with which a VACUUM ANALYZE is executed on all geometry columns (PostgreSQL-specific)

Table 20: Overview of all files within 3dcitydb folder.

3.3.1 Oracle

Step 1 - Define a user for the 3D City Database

A dedicated database user should be created for your work with the 3D City Database. This user must have the roles CONNECT and RESOURCE assigned and must own the privileges CREATE SEQUENCE and CREATE TABLE. If you plan to do a migration from the 3D City Database v2.1 (see chapter 3.4), the privilege CREATE SYNONYM is also necessary.

Note: The privileges CREATE SEQUENCE and CREATE TABLE are required for enabling and disabling spatial indexes. It is *not sufficient* to inherit these privileges through a role.

Step 2 – Connect to the Oracle database via SQL*Plus or SQL Developer

The script for setting up the database can be invoked from the command line tool *SQL*Plus* or from the *Oracle SQL Developer* software. For *SQL*Plus*, first start a shell environment of your preferred operating system and navigate to the directory where the CREATE_DB script is located. Then, a connection to the Oracle database is established, for instance, with the following command using the ezconnect functionality of *SQL*Plus*:

```
sqlplus [USER]/[PASS]@\"[HOSTNAME]:[PORT]/[SID]\"
```

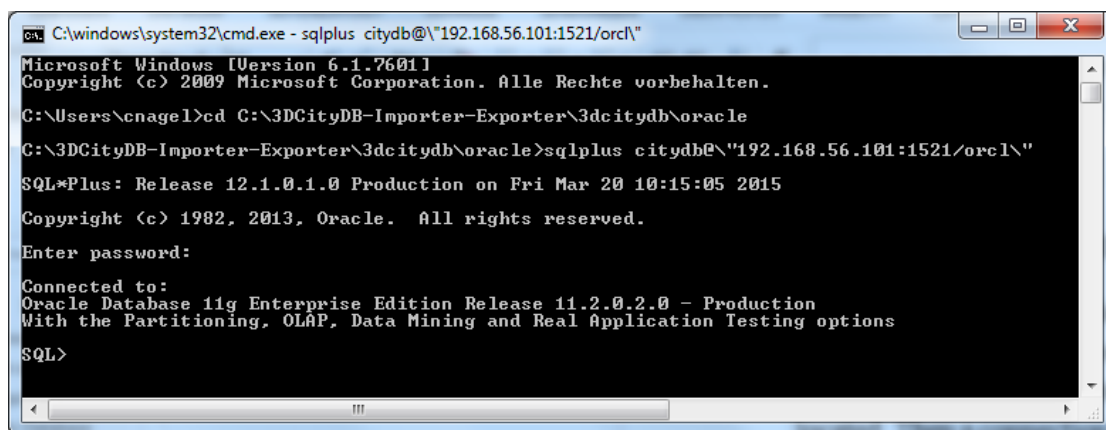


Figure 50: Connecting to Oracle using SQL*Plus and ezconnect.

Note: If you use a *SQL*Plus* client prior to version 11, ezconnect may not be available. In this case, you have to edit the *tnsnames.ora* file in order to specify the connection parameters. Please refer to your Oracle documentation for details.

Step 3 - Execute the CREATE_DB script:

Once the connection to the Oracle database is established, the setup script can be executed with the following command at the SQL prompt:

```
SQL> @CREATE_DB.sql;
```

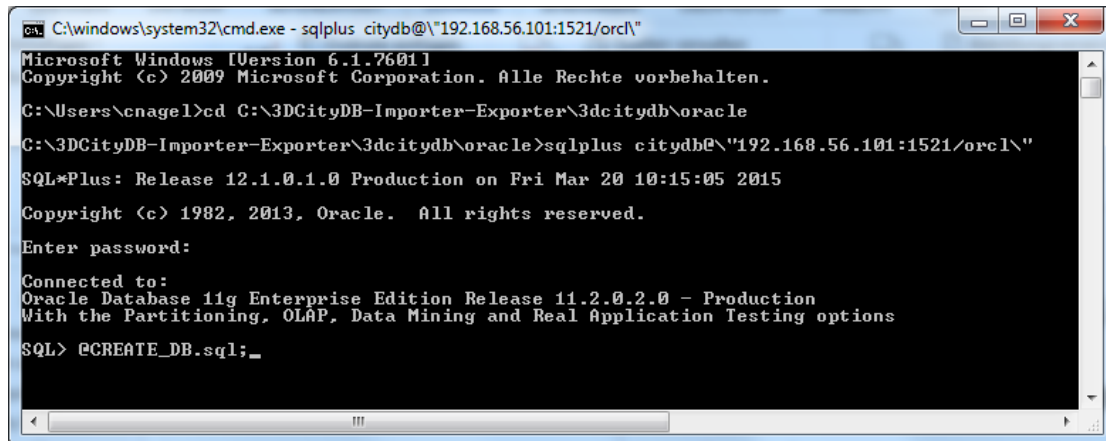


Figure 51: Command to execute the CREATE_DB.sql script.

Note: When using SQL Developer, directly execute CREATE_DB2.sql. Under the menu item 'Tools > Preferences > Database > Worksheet > Default path to look for scripts', enter the path to the CREATE_DB2.sql script. Then call the script from within an SQL sheet with the command @CREATE_DB2.sql.

Step 4 - Specify the coordinate reference system

After executing the CREATE_DB script, the user is prompted for the coordinate reference system (CRS) that shall be assigned to the geometry columns of the database. First, enter the Oracle-specific SRID (spatial reference ID) of the CRS. In most cases, this value will be identical to the EPSG code of the CRS. Second, provide the GML-conformant uniform resource name (URN) encoding of the CRS. More information about the SRID and the URN encoding can be found in chapter 2.3.4.

For both input parameters, default values are provided in parentheses at the user prompt. You can adopt the default values by pressing the *Enter* key without providing another value. Note that the default SRID value (81989002 for the Berlin Soldner CRS) is not available in Oracle per default. Please execute the script SOLDNER_BERLIN_CRS.sql in the subfolder UTIL/SRS (see Table 20) as SYS_DBA user before running CREATE_DB to make it available.

Step 5 – Enable or disable versioning

After providing the CRS information, the user is asked whether or not the database should be versioned-enabled. Versioning is realized based on Oracle's *Workspace Manager* functionality (see the Oracle documentation for more information). Please enter 'yes' or 'no'. The default value 'no' is confirmed by simply pressing *Enter*. Note that, in general, insert, update, delete

and index operations on version-enabled tables *take considerably more time* than on tables without versioning support.

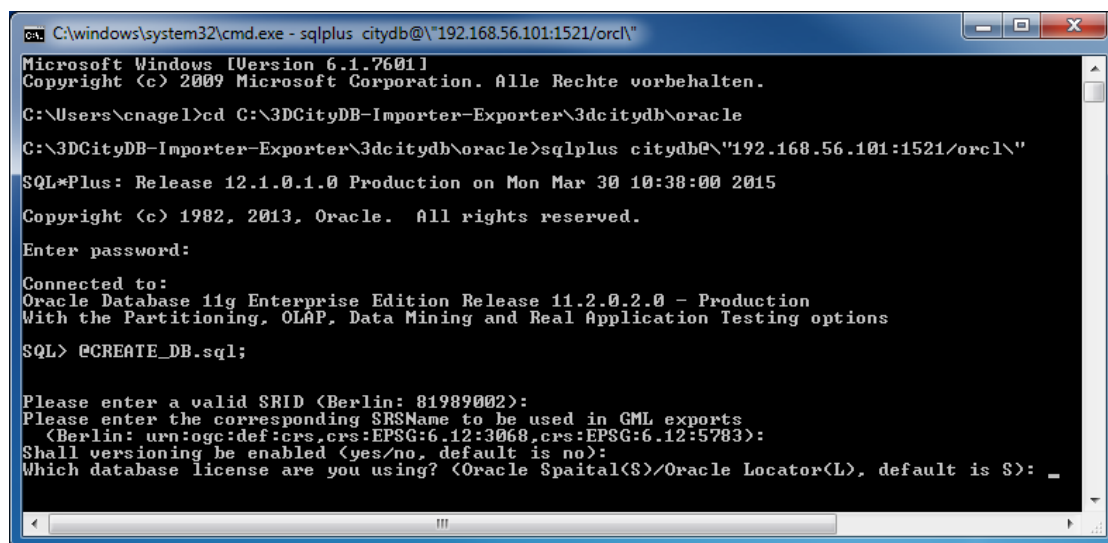
Step 6 – Choose Spatial or Locator license option

You can set up a 3D City Database instance on an Oracle database with *Spatial* or *Locator* support. Since *Locator* differs from *Spatial* with respect to the available spatial data types, you need to specify which license option is valid for your Oracle installation. Simply enter ‘L’ for *Locator* or ‘S’ for *Spatial* (default value) to make your choice.

Note: Since *Locator* lacks the GeoRaster data type, the 3D City Database tables for storing raster reliefs (RASTER_RELIEF, GRID_COVERAGE, GRID_COVERAGE_RDT) are not created when choosing *Locator*.

Note: Several spatial operations and functionalities that are available in Oracle *Spatial* are not covered by the *Locator* license even though they might be available from your Oracle installation. It is the **responsibility of the database user** to observe the Oracle license option. Choosing *Locator* or *Spatial* when setting up the 3D City Database does neither affect the license option nor the users’ responsibility.

The following figure exemplifies the required user input during steps 4 to 6.



```

C:\windows\system32\cmd.exe - sqlplus citydb@"192.168.56.101:1521/orcl"
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\cnagel>cd C:\3DCityDB-Importer-Exporter\3dcitydb\oracle
C:\3DCityDB-Importer-Exporter\3dcitydb\oracle>sqlplus citydb@"192.168.56.101:1521/orcl"
SQL*Plus: Release 12.1.0.1.0 Production on Mon Mar 30 10:38:00 2015
Copyright (c) 1982, 2013, Oracle. All rights reserved.
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> @CREATE_DB.sql;

Please enter a valid SRID (Berlin: 81989002):
Please enter the corresponding SRSName to be used in GML exports
(Berlin: urn:ogc:def:crs:crs:EPSG:6.12:3068,crs:EPSG:6.12:5783):
Shall versioning be enabled (yes/no, default is no):
Which database license are you using? (Oracle Spatial(S)/Oracle Locator(L), default is S): _

```

Figure 52: Example user input when executing CREATE_DB on an Oracle database.

Step 7 – Check if the setup is correct

After successful completion of the setup procedure, 62 tables, 13 sequences and 6 packages (that contain stored procedures) should appear in the user schema (Oracle Spatial).

The coordinate reference system specified during setup can be changed at any time. Execute the procedure `change_schema_srid` from the `GEODB_SRS` package that applies the procedure `change_column_sird` against all spatial columns within a user schema.

```
SQL> exec citydb_srs.change_schema_srid(  
    32632,  
    'urn:ogc:def:crs,crs:EPSG::25832,crs:EPSG::4326',  
    1);
```

The new EPSG code and URN encoding are specified through the first two parameters. The third parameter indicates whether the spatial data should be reprojected (=1) or not (=0). If a coordinate transformation is needed because an alternative reference system shall be used, the value '1' should be passed to the procedure. If a wrong SRID had been chosen by mistake during setup, a coordinate transformation might not be necessary in case the coordinate values of the city objects are already matching the new reference system. Thus, the value 0 should be provided to the procedure, which then only changes the spatial metadata to reflect the new reference system. It can also be omitted, as 0 is the default value for the procedure.

Versioning of the database can also be switched on and off at any time. The corresponding scripts are `ENABLE_VERSIONING.sql` and `DISABLE_VERSIONING.sql`. These scripts invoke routines of the Oracle Workspace Manager and will take some time for execution depending on the amount of data stored in the 3D City Database instance.

Last but not least, the schema and stored procedures of the 3D City Database can be dropped with the `DROP_DB` script, which is executed like `CREATE_DB`. Similar to `CREATE_DB`, you need to provide the license option (*Locator* or *Spatial*). Note that the script will *delete all data* stored in the 3D City Database schema. The database user will however not be deleted.

3.3.2 PostgreSQL

Step 1 - Create an empty PostgreSQL database

Choose a superuser or a user with the `CREATEDB` privilege to create a new database on the PostgreSQL server (e.g. 'citydb_v3'). The easiest way to create a new database is to use the graphical database client *pgAdmin* that is shipped with PostgreSQL. Please check the *pgAdmin* documentation for more details.

Important: As owner of this new database, choose or create a user who will later set up the 3D City Database instance. Otherwise, more permissions have to be granted. In the following steps, this user is called 'citydb_user'.

Step 2 – Add the PostGIS extension

The 3D City Database requires the PostGIS extension to be added to the database. This can **only be done as superuser**. The extension is added with the following command (or, alternatively, using *pgAdmin*):

```
CREATE EXTENSION postgis;
```

Step 3 - Execute the `CREATE_DB` script

The `CREATE_DB.sql` script can only be executed with PostgreSQL's *psql* client from a command line interface. It cannot be executed using *pgAdmin*. Therefore, open a shell

environment on your preferred operating system and navigate to the directory where the CREATE_DB script is located. Afterwards, execute the following command:

```
psql -h your_host_address -p 5432 -d name_of_database -U
your_username -f CREATE_DB.sql
```

An example session is presented in the following figure.

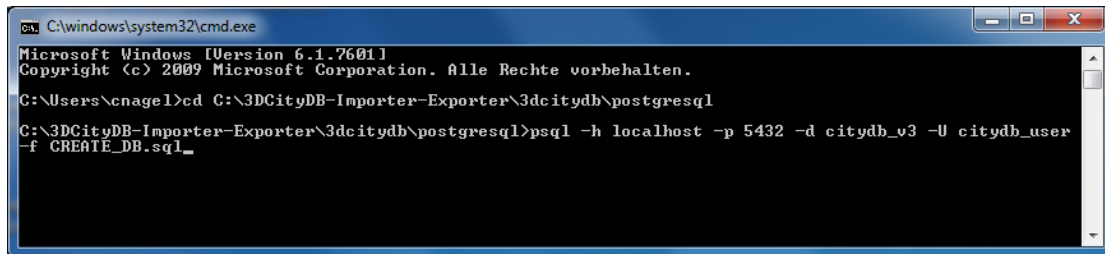


Figure 53: Executing the CREATE_DB.sql script in a shell using *psql*.

For convenience, a batch script (CREATE_DB.bat for Windows, CREATE_DB.sh for UNIX/Linux/MacOS) is provided that automatically starts the psql client in the background and runs the CREATE_DB script. However, **environment variables within the script must first be edited using a text editor**. These variables define the connection parameters to the database created in step 1 as well as the location of the psql executable. For example:

```
set PGPORT=5432
set PGHOST=localhost
set PGUSER=citydb_user
set CITYDB=citydb_v3
set PGBIN=C:\PostgreSQL\9.4\bin
```

The following figure exemplifies the contents of the CREATE_DB.bat file on Windows. Double-clicking on CREATE_DB.bat will start the setup procedure.

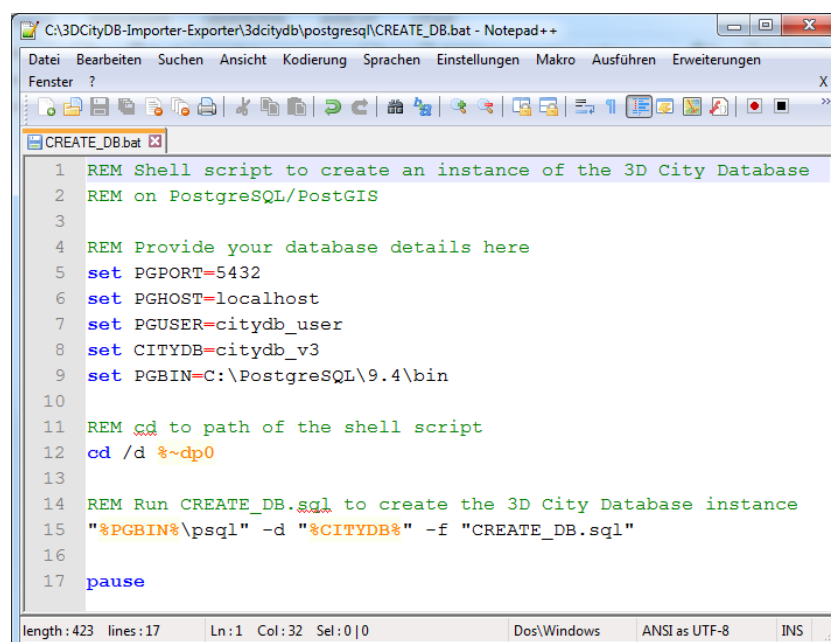


Figure 54: Using the CREATE_DB.bat script

On some UNIX/Linux distributions, you will have to run the CREATE_DB.sh script from within a shell environment. Please open your favorite shell and check whether execution permission is set for the starter script. Change to the installation folder and enter the following to make the starter script executable for the owner of the file:

```
chmod u+x CREATE_DB.sh
```

Afterwards, simply run the starter script by typing:

```
./CREATE_DB.sh
```

Step 4 – Specify the coordinate reference system

Like with the Oracle version, the user is prompted at the beginning to enter the SRID and the URN encoding of the coordinate reference system to be used (see chapter 2.3.4. for more information).

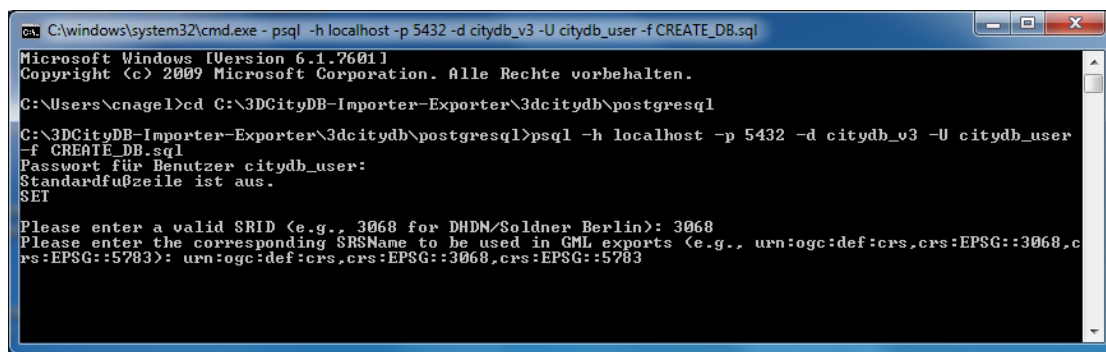


Figure 55: Example user input when executing CREATE_DB on a PostgreSQL database.

Note: Unlike the Oracle version, it is not possible to define default variables when prompting the user for input. Therefore, the values displayed in parentheses are just examples. The **user has to type in** the SRID value and the URN encoding and thus **cannot just continue by pressing Enter**.

Note: The setup process will terminate immediately if an error occurs. Reasons might be:

- The user executing CREATE_DB.sql is neither a superuser nor the owner of the specified database (or does not own privileges to create objects in that database);
- The PostGIS extension has not been installed; or
- Parts of the 3D City Database do already exist because of a previous setup attempt. Therefore, make sure that the schemas 'citydb' and 'citydb_pkg' do not exist in the database when setting up the 3D City Database.

After a series of log messages reporting the creation of database objects, the chosen reference system is applied to the spatial columns (except for those that will store data with local coordinate systems). This takes some seconds and is finished when the word 'Done' is displayed.

Step 5 – Check if the setup is correct

The 3D City Database is stored in a separate PostgreSQL schema called 'citydb'. It should contain 60 tables and 11 sequences. The stored procedures (count 101) are written to a separate PostgreSQL schema called 'citydb_pkg' that also contains one additional table and sequence. Usually different schemas have to be addressed in every query via dot notation, e.g.

```
SELECT * FROM citydb.building;
```

Fortunately, this can be avoided when the corresponding schemas are on the database *search path*. The search path is automatically adapted during the setup. Execute the command `SHOW search_path;` to check if the schemas citydb, citydb_pkg and public (for PostGIS elements) are contained.

Note: When using the created 3D City Database as a template database for new databases, the search path information is not transferred and thus has to be set again, e.g.:

```
ALTER DATABASE new_citydb_v3 SET search_path TO citydb,  
citydb_pkg, public;
```

The search path will be updated upon the next login, not within the same session.

To change the reference system after setting up the 3D City Database, use the command (see Oracle version for a description of the input parameters):

```
SELECT citydb_pkg.change_schema_srid(  
    32632,  
    'urn:ogc:def:crs,crs:EPSG::32632,crs:EPSG::4326',  
    1);
```

To drop the 3D City Database with all data, execute the DROP_DB.sql script in the same way like CREATE_DB.sql. Simply dropping the schemas 'citydb' and 'citydb_pkg' in a cascading way will also do the job.

3.4 Upgrade from previous version of the 3D City Database

If the previous version 2.1 of the 3D City Database is already in use, the v3.0.0 release offers a convenient way to migrate the data to the new database schema both in Oracle and PostgreSQL. Change to the folder 3dcitydb/[oracle_spatial/postgis]/MIGRATION from the installation directory.

Another safe and easy approach is to export the database content from the 2.x instance as CityGML with the previous version of the Importer/Exporter and to re-import the data into the new 3D City Database version by using the new Importer/Exporter shipped with this release. This approach might take more time though, depending on the amount of data stored in the database.

3.4.1 Oracle

Step 1 – Upgrade an existing installation

The migration to v3.0.0 *must be carried out on a version 2.1.0 instance* of the 3D City Database. Versions prior to version 2.1.0 must first be upgraded to 2.1.0 since the internal storage of envelopes of city objects changed substantially. Corresponding upgrade scripts are shipped with the v2.1.0 release. Upgrades to 2.1.0 can be carried out from any older version 2.0.0 to 2.0.6. A more detailed description of the upgrade procedure can be found in the document “Documentation of the 3D City Database v2.1.0 and the Importer/Exporter v1.6.0”.

Before upgrading your 3D City Database, a database backup is highly recommended to secure all data. The latter can be easily done using the Importer/Exporter tool or by tools provided by Oracle.

Important: Please note that the last step in the upgrade process is a lengthy one. Altering the internal storage of the envelopes of all city objects in a large and/or versioned database may take hours. Depending on their initial state, spatial indexes may be disabled and re-enabled in the process, adding to the duration as a whole. This process **MUST NOT** be interrupted since it could lead to an inconsistent state. Please be patient and remember that backing up all of your data before starting any database upgrade is the commonly recommended practice.

File	Explanation
GRANT_ACCESS	Grant SELECT on v2.1.0 schema to user with 3.0.0 schema
MIGRATE_DB	Top level script to start the migration
MIGRATE_DB_V2_V3	PL/SQL package that contains the migration logic
MIGRATE_DB_V2_V3_Sptl	PL/SQL procedure to migrate raster data (only needed for Oracle Spatial)

Table 21: Overview of migration scripts in Oracle version

Step 2 – Creating a new installation

The migration script transfers data from a user schema with the v2.1.0 installation to another user schema that has to contain the 3D City Database schema v3.0.0. Install the new version like it is described in chapter 3.3.1 if not happened yet.

Step 3 – Grant select on v2.1.0 schema to v3.0.0 schema

For the migration process, it is necessary that the user with the v3.0.0 schema can access the user schema with the v2.1.0 version. Therefore, connect to the schema with 3D City Database v2.1.0 and run the script GRANT_ACCESS from the 3dcitydb/oracle_spatial/MIGRATION folder within the installation directory of the Import/Export tool. When executed the user is requested to type in the schema name for the 3D City Database v3.0.0 instance.

Step 4 – Run MIGRATE_DB

Now, connect to the schema that contains the 3D City Database v3.0.0 and run the script MIGRATE_DB (same folder like GRANT_ACCESS). This time the user is requested to specify the name of the schema with the v2.1.0 instance.

Step 5 – Choose Spatial or Locator license option

With the second input parameter specify the database license running on your Oracle server, like you have done when setting up the v3.0.0 instance of the 3D City Database. Choose 'S' for Spatial (which will additionally migrate raster data) and 'L' for Locator.

Step 6 – Be sure of using unique texture URIs

In the new version of the 3D City Database, textures that are referenced to more than one geometry are no longer stored redundantly in the SURFACE_DATA table but only once in the TEX_IMAGE table. This optimization can also be done during the migration process, if it is guaranteed that texture URIs are unique and not used for different texture files. Otherwise, some textures would get lost during the migration and remaining images would be referenced to wrong surfaces. Therefore, as for the third input parameter of the MIGRATE_DB script the user is asked, if he can assure the non-existence of duplicate texture URIs, which he can verify with 'y' or 'yes'.

In case you know that textures in the database are named equally (or if you do not know) you can still run the script by entering 'n' or nothing (because it is the default). Entries in the TEX_IMAGE column of the SURFACE_DATA table from version 2.1 are then further mapped 1:1 to the new TEX_IMAGE table of version 3.0.0.

Note: A simple unification of texture URIs in advance of the migration will not help to store the textures only once, because same textures with different URIs are regarded as different image files and would all end up in the new TEX_IMAGE table. You would have to compare the binary data itself.

Step 7 – Check if the setup is correct

The script temporarily creates a set of synonyms for the tables of the v2.1.0 schema, an additional package with migration procedures as well as functions. The elements are removed again when the migration progress is completed and the message "DB migration is completed successfully." is displayed on the console. It is recommended to generate a database report of the new user schema and compare it with a report of the schema that contains the 2.1 instance of the 3D City Database (done with the previous version of the Import/Export tool).

3.4.2 PostgreSQL

Step 1 – Run MIGRATE_DB

The migration of a 3D City Database instance v2.x towards v3.0.0 works very easy under PostgreSQL. Setting up a new v3.0.0 instance is not necessary. Simply execute the script MIGRATE_DB within the folder 3dcitydb/postgis/MIGRATION and specify the already existent database with the v2.1.0 schema (very similar to CREATE_DB):

```
psql -h your_host_address -p 5432 -d name_of_database -U
your_username -f MIGRATE_DB.sql
```

The user is free to use the batch script MIGRATE_DB.bat (Windows) or MIGRATE_DB.sh (Linux/Unix) instead. Like with the CREATE_DB script connection parameters have to be edited within a text editor.

File	Explanation
CONSTRAINTS_V3.sql	Adds constraints to the v3.0.0 schema (executed at the end of the migration process)
CREATE_DB_V3.sql	Creates elements of the 3D City Database schema that are new since v3.0.0
DROP_DB_V2.bat	Batch file that calls DROP_DB_V2.sql (for MS Windows family)
DROP_DB_V2.sh	Shell script that calls DROP_DB_V2.sql (UNIX/ Linux & derivations, MacOS X)
DROP_DB_V2.sql	Deletes all elements of the old 3D City Database relational schema of v2.x
FUNCTIONS	Contains functions <ul style="list-style-type: none"> to transform a string of texture coordinates into a polygon of type GEOMETRY and a procedure to update sequences to help filling the city object table
INDEXES_V3	Adds indexes to the v3.0.0 schema (executed at the end of the migration process)
MIGRATE_DB.bat	Batch file that calls MIGRATE_DB.sql (for MS Windows family)
MIGRATE_DB.sh	Shell script that calls MIGRATE_DB.sql (UNIX/ Linux & derivations, MacOS X)
MIGRATE_DB.sql	Top level script to start the migration
MIGRATE_DB_V2_V3.sql	SQL script that contains the migration logic

Table 21: Overview of migration scripts in PostgreSQL version

Important notice if using PostgreSQL pre-9.3. If you are using a PostgreSQL version lower than 9.3, the script will run into an error just after starting. Please open the MIGRATE_DB script within a text editor and change the lines 30 to 36 like shown in the following:

```
-- SELECT srid FROM database_srs \gset
-- alternative way for PostgreSQL versions pre 9.3 that have no
\gset command
\echo 'Database SRID:'
SELECT srid FROM database_srs;
\prompt 'Please enter the EPSG code of the SRID used in the
current database: ' srs_no
\set srid :srs_no
```

Re-run the script. Now the user is requested to type in the SRID of the reference system that is used for the 3D City Database instance. It is also displayed in the console as a little help.

Step 2 – Be sure of using unique texture URIs

Like with the Oracle version, you are requested to guarantee that no texture URI is used for different images. See Step 6 in the workflow explanation of the Oracle version for further details.

Step 3 – Check if the setup is correct

After a series of log messages reporting the selection of data from the v2.x schema, updates of references and the creation of database objects, the script is finished with the message '3DCityDB migration complete!'. If the old database schema is not dropped during the migration (see next chapter), both versions of the 3D City Database will remain in one database. This is actually a good thing, because you can further compare if everything has been transferred correctly.

Rerun the migration

If the migration process has been interrupted by the user or by severe software errors, the migration script can simply be executed again (only if the old v2.x schema still exists) without manually dropping already created parts of the v3.0.0 schema because the script does it for you.

3.4.3 Dropping the old v2.x schema (Oracle and PostgreSQL)

If using PostgreSQL, open the script MIGRATE_DB within a text editor and move to lines 79 to 82. Dropping the old v2.1.0 schema can also be done during the migration. Simply remove the double hyphens.

However, it is recommended to first complete the migration process and check if the content has been transferred correctly. Take use of the functions of the Import/Export tool to verify that

- no city objects are missing (do a database report),
- relations between features and attributes are correct, and
- exports look correct inside a viewer application.

For Oracle users dropping the old schema can then be done like explained in chapter 3.3.1 or by deleting the user. PostgreSQL users **MUST NOT** execute the DROP_DB script of version 2.x as the old and new instance of the 3D City Database are both stored inside the same database (new = citydb schema, old = public schema). The script does not distinguish between schemas and would drop tables of the citydb schema, as it comes first in the search path of the database. **Use instead the script DROP_DB_V2** from inside the MIGRATION folder.

4 Stored procedures and additional features

The 3D City Database is shipped with a set of stored procedures referred to as the CITYDB package (formerly known as the GEODB package). They are automatically installed during the setup procedure of the 3D City Database. For the Oracle version, it comprises of six PL/SQL packages. In the PostgreSQL version, functions are written in PL/pgSQL and stored in their own database schema called ‘citydb_pkg’.

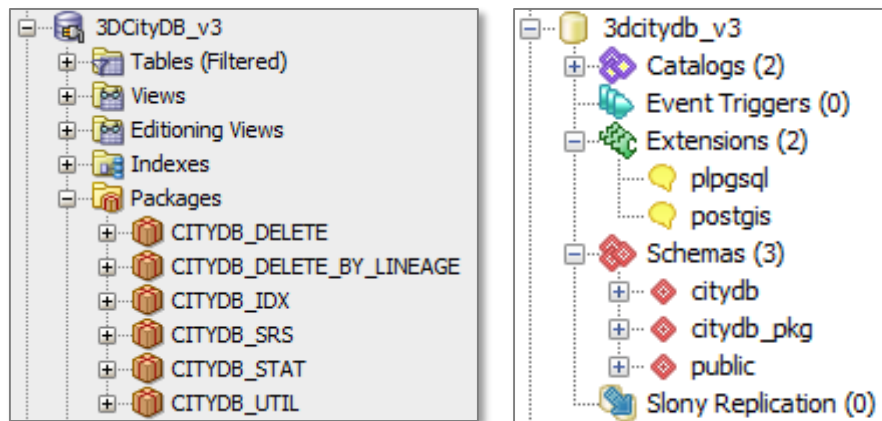


Figure 56: Graphical database client connected to the 3D City Database (left: SQL Developer (Oracle), right: pgAdmin (PostgreSQL))

Many of these functions and procedures expose certain tasks on the database side to the Importer/Exporter client. For detailed information about individual functions, please refer to the comments inside the corresponding SQL files in the 3dcitydb/oracle_spatial/PL_SQL or 3dcitydb/postgis/PL_pgSQL folder of the installation directory.

When calling stored procedures, the package name has to be included for the Oracle version. With PostgreSQL, the ‘citydb_pkg’ schema has not to be specified as prefix since it is put on the database *search path* during setup. The following examples return the version number of the installed 3D City Database instance by invoking the function `citydb_version` from the CITYDB_UTIL package:

Oracle:

```
SELECT * FROM TABLE(citydb_util.citydb_version);
```

PostgreSQL:

```
SELECT * FROM citydb_version();
```

Almost every stored procedure is schema sensitive in that it allows the user to specify a *database schema* (as last input parameter) that is bound to query strings inside the function body. The default schema name in Oracle is the user name (system variable `USER`) and in PostgreSQL it is ‘citydb’. For Oracle, it would be possible to execute stored procedures against external user schemas with an installed 3D City Database instance if sufficient privileges are supplied. In PostgreSQL, they could be run against any schema of a database that contains the 3D City Database instance. This can be useful when maintaining alternate version of a city model in different schemas (e.g. planning scenarios, different timestamps).

4.1 CITYDB_UTIL

In the Oracle version, the script of the package `CITYDB_UTIL` does also contain definitions of data types that are used by functions from other packages (these are not necessary in PostgreSQL), e.g.

- `STRARRAY`, a nested table of the data type `VARCHAR2`
- `ID_ARRAY`, a nested table of the data type `NUMBER`
- `DB_VERSION_OBJ`, an object that bundles version information of the installed 3D City Database instance
- `DB_VERSION_TABLE`, a nested table of `DB_VERSION_OBJ`
- `DB_INFO_OBJ`, an object that bundles metadata of the used reference system
- `DB_INFO_TABLE`, a nested table of `DB_INFO_OBJ`

Most of the functions of this package serve a very simple task and are not explained in detail here. One notable procedure is `update_schema_constraints`. It modifies all foreign keys on each table in the schema (`update_schema_constraints`) and sets them to `ON DELETE CASCADE` instead of the default `ON DELETE RESTRICT`. No value has to be passed to the procedure as `'CASCADE'` is the default value. Any other character sequence would set the constraints back to `ON DELETE RESRTICT`.

Cascading deletes can slightly improve the performance of deletion processes because foreign keys are used to delete referencing entries in other tables. The delete functions of the 3D City Database (see chapter 4.5) are programmed along the references and not against them. Thus a lot more functions and queries have to be executed.

4.2 CITYDB_IDX

The package `CITYDB_IDX` provides functions to create, drop, and check both spatial and non-spatial indexes on tables of the 3D City Database by using a user-defined data type called `INDEX_OBJ`. In the Oracle version, the data type offers three member functions to construct an `INDEX_OBJ`. In the PostgreSQL version, these are just separate functions within the `'citydb_pkg'` schema:

- `construct_spatial_3d` for a 3-dimensional spatial index
- `construct_spatial_2d` for a 2-dimensional spatial index
- `construct_normal` for a normal B-tree index

The easiest way to take use of this package is by using the Importer/Exporter (see chapter 5.2.2), which provides an interface for enabling and disabling indexes (ON and OFF). Disabling spatial indexes can accelerate some operations such as bulk imports, deletion of many objects, and migration of data from a 3D City Database v2.1.0 instance to version 3.0.0.

The list of indexes can be extended in the table `INDEX_TABLE` (part of the `'citydb_pkg'` schema in PostgreSQL).

4.3 CITYDB_SRS

The package `CITYDB_SRS` provides functions and procedures dealing with the coordinate reference system used for the 3D City Database instance, for instance, to check if a 3D reference system is used (`is_db_coord_ref_sys_3d`). The most essential procedure `change_schema_srid` has already been explained in the installation chapter. The function `transform_or_null` is applied during CityGML exports if the data shall be transformed into another coordinate system for the output document (see chapter 5.4).

4.4 CITYDB_STAT

The package `CITYDB_STAT` currently only serves a single purpose: To count all entries in all tables and generate a report as an array of string values (`STRARRAY` data type in Oracle, `text[]` in PostgreSQL). The tabulator escape sequence `\t` is used to generate a nice looking report for the Importer/Exporter.

4.5 CITYDB_DELETE and CITYDB_DELETE_BY_LINEAGE

The package `CITYDB_DELETE` consists of several functions that facilitate to delete single city objects. Each function automatically takes care of integrity constraints between relations in the database. The package is meant as low-level API providing a delete function for each relation (except for linking tables) – from a single polygon in the table `SURFACE_GEOMETRY` (`delete_surface_geometry`) up to a complete city object (`delete_cityobject`) or even a whole city object group (`delete_cityobjectgroup`). This should help users to develop more complex delete operations on top of these low-level functions without re-implementing their functionality.

In order to illustrate the low-level approach of this package, assume a user wants to delete a building feature together with all its nested subfeatures. For this purpose, the user calls the `delete_building` (or `delete_cityobject`) function, which internally leads to subsequent calls to the following stored procedures:

- `delete_building` for the building and its dependent building parts (recursive call)
- `delete_thematic_surface` for dependent boundary surfaces of the building (nested call of `delete_opening` for dependent openings of the boundary surfaces)
- `delete_building_installation` for dependent outer installations of the building (nested call of `delete_thematic_surface` for boundary surfaces of the installations)
- `delete_room` for dependent rooms of the building (nested call of `delete_thematic_surface` for interior boundary surfaces, `delete_building_installation` for interior installation and `delete_building_furniture` for furniture within the room)
- `delete_address` for dependent addresses that are not referenced by other buildings and bridges

- `intern_delete_surface_geometry` (`delete_surface_geometry` in PostgreSQL) for deleting the geometry of the building and its nested features
- `intern_delete_cityobject` to remove the entry in the `CITYOBJECT` table that corresponds to the deleted building (also deletes generic attributes, external references, appearances, etc.)

Most of the stored procedures take the *primary key id* value (PID) of the entry to be deleted as input parameter and return the id value (or a set of id values) if the entry has been successfully removed. Some functions offer an optional second flag parameter to indicate whether additional processes shall be executed (1 = true) or not (0 = false).

For the functions `delete_surface_geometry` and `delete_implicit_geometry`, it can be decided whether referencing appearances shall be deleted as well (`CLEAN_APPS = 1`, which executes `cleanup_appearances`) or if the reference will only be cut off by just deleting entries in the `TEXTUREPARAM` table (`CLEAN_APPS = 0`) (default is false). As one appearance can be assigned to several polygons, it has to be checked whether the appearance is not further referenced by another geometry element. When deleting many entries from `SURFACE_GEOMETRY` this check would be very time consuming when performed for each iteration. Therefore, `cleanup_appearances` should be performed afterwards.

The same principle applies to other cleanup functions when deleting multiple city objects (note that the third parameter `CLEANUP` of `delete_cityobject` is 0), e.g.

Oracle:

DECLARE

```
pid number;
dummy_ids ID_ARRAY := ID_ARRAY();
```

BEGIN

```
FOR rec IN (SELECT * FROM cityobject WHERE ...) LOOP
    pid := citydb_delete.delete_cityobject(rec.id, 0, 0);
END LOOP;
dummy_ids := citydb_delete.cleanup_implicit_geometries(1);
dummy_ids := citydb_delete.cleanup_appearances(1);
dummy_ids := citydb_delete.cleanup_cityobjectgroups;
dummy_ids := citydb_delete.cleanup_citymodels;
```

END;

PostgreSQL:

```
SELECT delete_cityobject(id, 0, 0) FROM cityobject WHERE ... ;
SELECT cleanup_implicit_geometries(1);
SELECT cleanup_appearances(1);
SELECT cleanup_cityobjectgroups();
SELECT cleanup_citymodels();
```


As can be seen in the above example, the function `cleanup_appearances` also takes a flag attribute called `ONLY_GLOBAL` as optional input parameter. If `ONLY_GLOBAL` is set to `true` (which is the default value), only global appearance information will be deleted. In the given example, it is sufficient to cleanup only global appearances because local ones have already been removed for each city object (through the function `intern_delete_cityobject`, see previous example).

The functions `delete_cityobjectgroup` and `delete_citymodel` both take a flag called `DELETE_MEMBERS`, which allows to delete all contained city objects. As a group also is a city object, the flag `DELETE_MEMBERS` is also available for the top-level function `delete_cityobject` as second parameter (see last example).

The function `delete_cityobject_cascade` is an alternative way to `delete_cityobject` as it does not call any other delete function except for cleanup functions, which is an optional step (flag parameter `CLEANUP`). **Important:** Executing `delete_cityobject_cascade` requires that foreign key constraints are set to `ON DELETE CASCADE`. This can be done with the function `update_schema_constraints` from the `CITYDB_UTIL` package (see chapter 4.1).

In the Oracle version, updating the constraints is also mandatory for using the procedure `cleanup_schema`, which removes all entries from all tables and resets the sequences. In the PostgreSQL version, changing the constraints this is not necessary as `cleanup_schema` works differently.

The `CITYDB_DELETE_BY_LINEAGE` package offers an API to delete multiple city objects that share a common value in the `LINEAGE` column of the `CITYOBJECT` table. If the object class is known to the user, a specific delete function can be chosen. Otherwise, the generic function `delete_cityobjects` should be used. A cleanup process is automatically triggered afterwards. `DELETE_MEMBERS` is available as an optional flag parameter.

5 Importer / Exporter

The *3D City Database Importer/Exporter* is a Java-based front-end for the 3D City Database and allows for high-performance loading and extracting 3D city model data.

The supported import and export operations are:

- Import of CityGML models (cf. chapter 5.3);
- Export data as CityGML models (cf. chapter 5.4);
- Export data in KML/COLLADA format (cf. chapter 5.5); and
- Export data as spreadsheets (available as plugin, cf. chapter 6.2).

Please refer to chapter 3.1 for system requirements and a documentation of the installation procedure.



The 3D City Database Importer/Exporter is free software under the *GNU Lesser General Public License version 3.0*. See the `LICENSE.txt` file shipped with the software for more details. For a copy of the GNU Lesser General Public, please visit <http://www.gnu.org/licenses/>.

5.1 Running and using the Importer / Exporter

The 3D City Database Importer/Exporter offers both a graphical user interface (GUI) and a command line interface (CLI). The CLI allows for embedding the tool in batch processing workflows and third party applications. The usage of the CLI is documented in chapter 5.8.

In order to launch the GUI, simply use the starter scripts located in the installation folder of the 3D City Database Importer/Exporter. A desktop icon as well as shortcuts in the start menu of your operating system will additionally be available in case you chose to create shortcuts during setup. Depending on your platform, one of the following starter scripts is provided:

- `3DCityDB-Importer-Exporter.bat` (Microsoft Windows family)
- `3DCityDB-Importer-Exporter.sh` (UNIX/Linux family)
- `3DCityDB-Importer-Exporter.command` (Mac OS X family)

On most platforms, double-clicking the starter script or its shortcut runs the Importer/Exporter.

For some UNIX/Linux distributions, you will have to run the starter script from within a shell environment though. Please open your favourite shell and first check whether execution rights are correctly set on the starter script. If not, change to the installation folder and enter the following command to make the starter script executable for the owner of the file:

```
chmod u+x 3DCityDB-Importer-Exporter.sh
```

Afterwards, simply run the software by issuing the following command:

```
./3DCityDB-Importer-Exporter.sh
```

Note: With every release, the `README.txt` file in the installation folder provides up-to-date and version-specific information on how to run the Importer/Exporter.

The starter scripts define default values for the Java Virtual Machine (JVM) that runs the Importer/Exporter. Most importantly, they specify the maximum available main memory for the application through the `-Xmx` parameter of the JVM. The default values have been chosen to be reasonable for most platforms but may need to be *adapted to your needs* before launching the application (e.g., if you want to increase the available main memory).

The graphical user interface of the Importer/Exporter is organized into four main components as shown in Figure 57. A *menu bar* [1] is located either below (Windows, some Linux distributions) or above (Mac, some Linux distributions) the title bar. The main application window is divided into an *operations window* [2] that renders the user dialogs of the separate operations of the Importer/Exporter and a *console window* [4] that displays log messages. Via the View entry in the menu bar, the console window can be detached from the main window and rendered in a separate window. At the bottom of the operations window, a *status bar* [3] provides information about running processes and database connections.

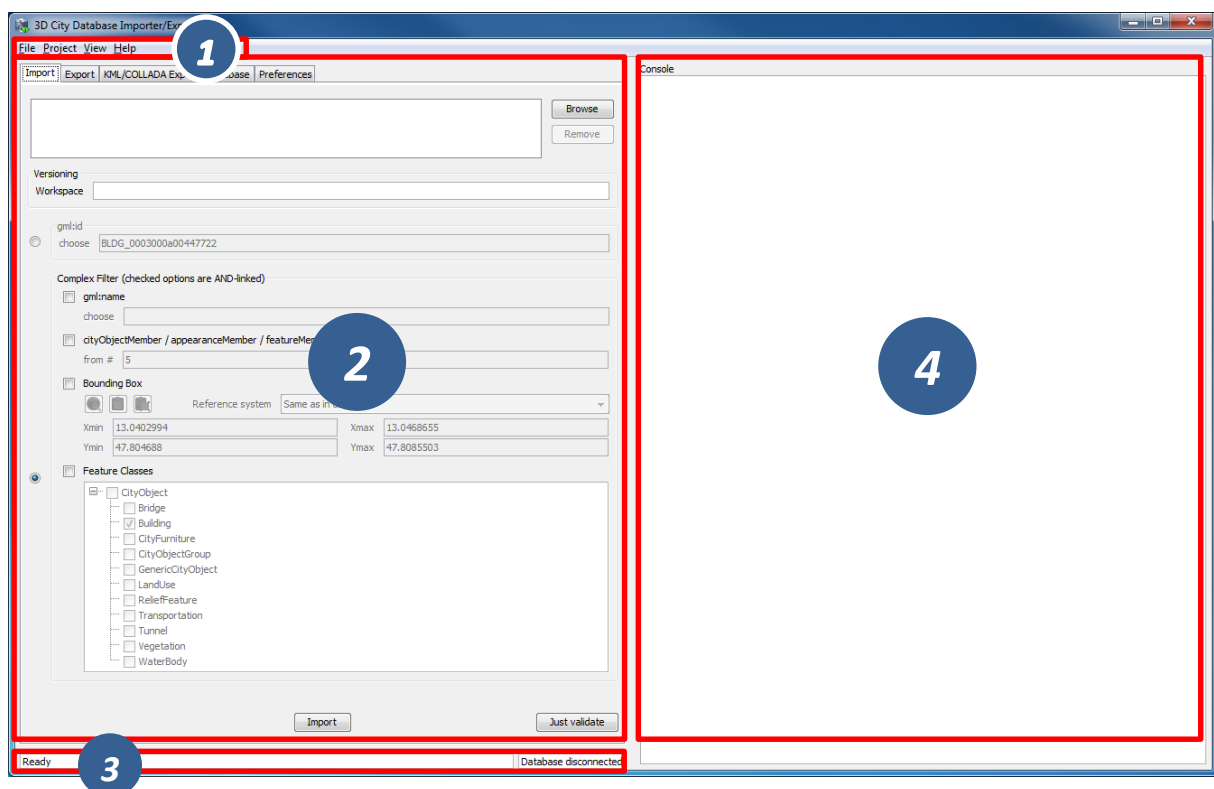


Figure 57: Organization of the Importer/Exporter GUI.

The tab menu on top of the operations window lets you switch between the operations of the Importer/Exporter and their user dialogs. The following tabs are available:

- | | |
|----------------------|---|
| • Import | Import of CityGML models into the database |
| • Export | Export of city model data as CityGML |
| • KML/COLLADA Export | Export of city model data in KML/COLLADA format |
| • Database | Database connection settings and operations |

- **Preferences** Preference settings for each operation

Note: If you have installed plugins, the tab menu may contain additional entries. Please refer to the documentation of your plugin in this case.

The main menu bar [1] offers the entries `File`, `Project`, `View` and `Help`. The `File` menu only contains one entry `Exit` to close the application.

The `Project` menu lets a user store and load settings from a config file. The separate menu entries provide the following functionality:

<i>Open Project...</i>	Load a config file and recover all settings from this file.
<i>Save Project</i>	Save all settings made in the GUI to the default config file.
<i>Save Project As...</i>	Save all settings made in the GUI to a separate config file.
<i>Restore Default Settings</i>	Set all settings to default values.
<i>Save Project XSD As...</i>	Save the XML Schema defining the XML structure of config files to a separate file. The XML Schema is helpful in case a user wants to manually edit the config file. Only config files conforming to the XML Schema definition will be successfully loaded by the Importer/Exporter.
<i>Recently Used Projects...</i>	List of recently loaded config files.

Note: The Importer/Exporter uses one *default config file* per operating system user running the Importer/Exporter. All settings made in the GUI are automatically stored to this default config file when the Importer/Exporter is closed and are recovered from this file upon program start. Thus, all user-defined settings are remembered. The default config file is named `project.xml` and is stored in the *home directory* of the user. Precisely, you will find the config file in the subfolder `3dcitydb/importer-exporter-3.0/config`. However, the location of the home directory differs for different operating systems. Using environment variables, the location can be identified dynamically:

- `%HOMEDRIVE%%HOMEPATH%\3dcitydb\importer-exporter-3.0\config` (Windows 7 and higher)
- `$HOME/3dcitydb/importer-exporter-3.0/config` (UNIX/Linux, Mac OS families)

The `View` menu affects the GUI elements of the Importer/Exporter and offers the following entries:

<i>Open map window</i>	Opens the 2D map window for bounding box selections (cf. chapter 5.7).
------------------------	--

Detach Console

Renders the console window in a separate application window.

Restore default perspective

Restores the GUI to its default settings.

Finally, the Help menu gives access to an Info dialog and the Read Me file shipped with the Importer/Exporter. Amongst other information, the Info dialog displays the official *version and build number* of the Importer/Exporter.

5.2 Database connections and operations

The Database tab of the operations window shown in the figure below allows a user to manage and establish database connections [1] and to execute database operations [2].

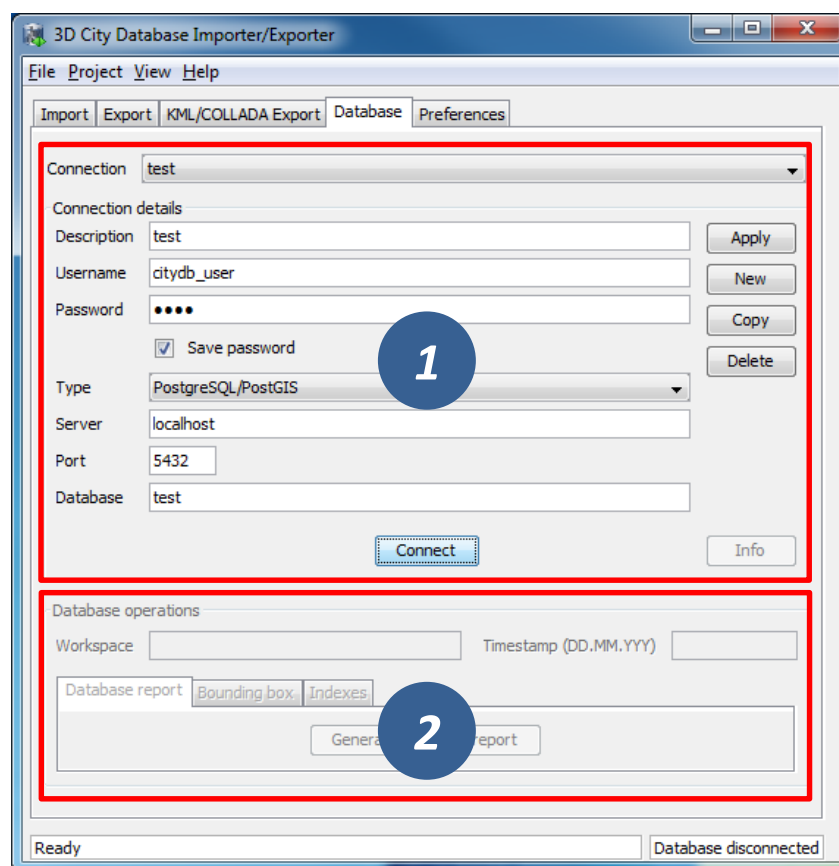


Figure 58: Database tab.

5.2.1 Managing and establishing database connections

In order to connect to an instance of the 3D City Database, valid connection parameters must be provided on the Database tab. In contrast to previous versions of the Importer/Exporter, this version can connect to both Oracle and PostgreSQL databases. The list of supported database systems may be extended in upcoming releases.

Database connection details comprise the *username* and *password* of the database user, the *type* of the database, the *server* name (network name or IP address) and *port* number (default: 1521 for Oracle; 5432 for PostgreSQL) of the database server, and the *database* name (when using Oracle, enter the database SID or service name here). If you need assistance, ask your database

administrator for this information. For convenience, a user can choose to *save the password* in the config file of the Importer/Exporter. Please be aware that the password will be stored as plain text.

In order to manage more than one database connection, connection details are assigned a short *description* text. The drop-down list at the top of the Database tab allows a user to switch between connections based on their description. By using the *Apply*, *New*, *Copy* and *Delete* buttons, edits to the parameters of the currently selected connection can be saved, a new connection with empty connections details can be created, and existing connections can be copied or deleted from the list.

The *Connect / Disconnect* button lets a user connect to / disconnect from a 3D City Database instance based on the provided connection details.

Note: With this version 3.0 of the Importer/Exporter, you will be able to **connect to version 3.x instances** of the 3D City Database **but not to any previous version**. Moreover, you will not be able to connect to version 3.x instances of the 3D City Database with an Importer/Exporter prior to version 3.0. See chapter 3.4 for a guide on how to migrate a version 2.x instance of the 3D City Database to the latest version 3.0.

The console window logs all messages that occur during the connection attempt. In case a connection could not be established, error messages are displayed that help in identifying the cause of the connection problem. Otherwise, the console windows contains information about the connected 3D City Database instance similar to those shown in Figure 59. This information comprises the version of the 3D City Database, the name and version of the underlying database system, the spatial reference system ID (SRID) as well as its name and GML encoding (as specified during the setup of the 3D City Database), and whether or not the database tables are version-enabled.

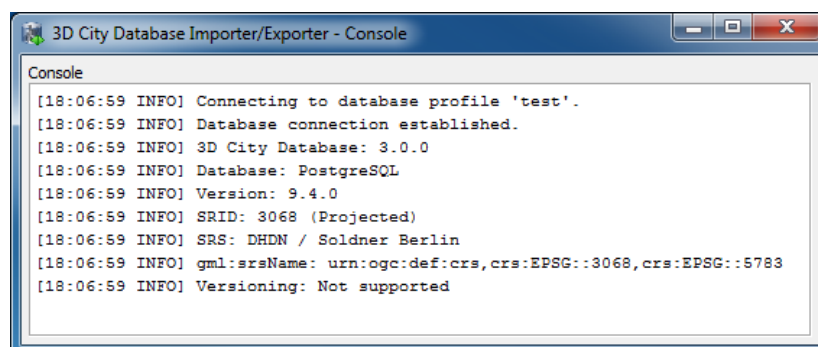


Figure 59: Log messages for a successful database connection.

This information can be requested from a connected 3D City Database at any time using the *Info* button on the Database tab. Upon successful connection, the description text of the connection is moreover displayed in the title bar of the application window.

5.2.2 Executing database operations

After having established a connection to an instance of the 3D City Database, the Database tab (cf. [2] in Figure 58) offers the following database operations to be executed on that instance:

- Generating a database report;
- Calculating the bounding box of selected feature types; and
- Managing indexes on database tables.

Generating a database report. A database report is a list of all tables of the 3D City Database together with their total number of rows. This operation therefore provides a quick overview of the contents of the 3D City Database. The report is printed to the console window.

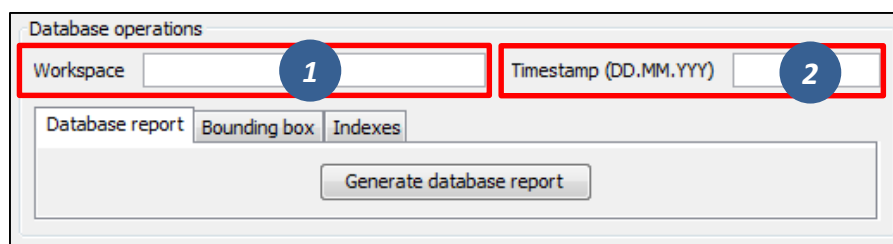


Figure 60: Generating a database report.

If the database is version-enabled (Oracle only), the database report can be created for the contents of a specific *workspace* [1] at a given *timestamp* [2]. If no workspace is specified, the default workspace is chosen as fallback (Oracle: *LIVE*). If the workspace does not exist, a corresponding error message is provided. Workspaces are not a feature of the 3D City Database but are managed through the *Oracle Workspace Manager* tool. Please refer to the Oracle database documentation for details. Since PostgreSQL does currently not support workspaces, the corresponding input fields are disabled when connecting to a 3D City Database running on PostgreSQL.

Calculating the bounding box. This operation calculates the 2D bounding box of the city objects stored in the database. The bounding box is useful, for instance, as input to spatial filters in CityGML imports and exports as well as KML/COLLADA exports (see documentation of the corresponding operations).

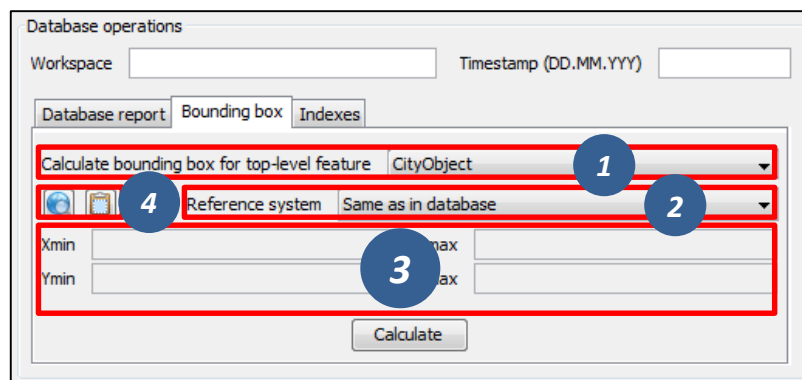




Figure 61: Calculating the bounding box for selected feature types.

The coordinate values of the lower left ($Xmin$, $Ymin$) and upper right ($Xmax$, $Ymax$) corner of the calculated bounding box are rendered in the corresponding fields of the dialog [3]. The values are also copied to the clipboard of your operating system and can therefore easily be pasted into the import and export dialogs. You can also manually copy the values to the clipboard by clicking the  button [4], or by right-clicking on a data field [3] and choosing the corresponding option from the context menu.

The calculation of the bounding box can be restricted to a specific city object type such as `Building` or `WaterBody` [1]. Similar to the generation of a database report, the user can request the bounding box for city objects living in a specific *workspace* at a given *timestamp* if the database is version-enabled (Oracle only). The coordinate values can optionally be transformed into a user-defined coordinate *reference system* that is available from the drop-down list [2]. Per default, the coordinate values are presented in the same reference system as specified for the 3D City Database instance during setup. See chapter 2.3.4 for details on how to define and manage user-defined reference systems.

By using the map button  [4], the calculated bounding box is rendered in a separate 2D map window for visual inspection as shown below. The usage of this map window is described in chapter 5.7.

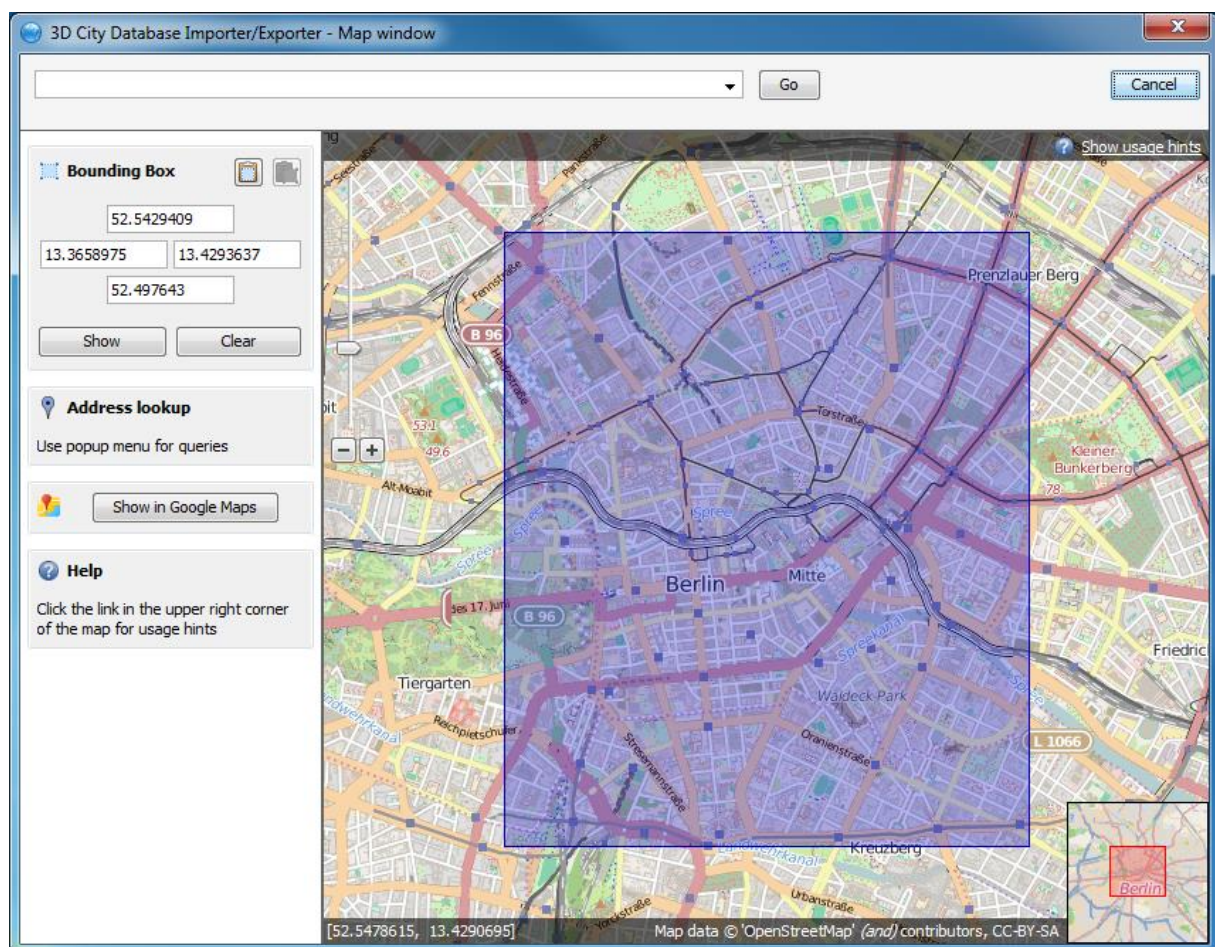


Figure 62: Map window for displaying and choosing bounding boxes. Note that the coordinate values of the bounding box are shown in the upper left component.

Managing indexes. The Importer/Exporter allows the user to manually activate or deactivate indexes on predefined tables of the 3D City Database schema, and to check their status.

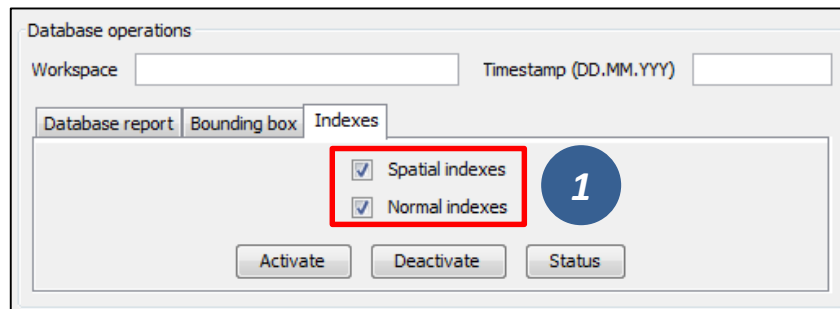


Figure 63: Managing spatial and normal indexes.

The operation dialog differentiates between *spatial indexes* on geometry columns and *normal indexes* on columns with any other datatype [1]. The buttons *Activate*, *Deactivate*, and *Status* trigger a corresponding database process on spatial indexes only, normal indexes only or both index types depending on which checkboxes are selected [1]. Again, the user can define a *workspace* and *timestamp* on which the operation shall be executed if the database is version-enabled (Oracle only).

The index operations only affect the following important subset of all indexes defined by the 3D City Database schema:

- *Spatial index* on column ENVELOPE of table CITYOBJECT
- *Spatial index* on column GEOMETRY of table SURFACE_GEOMETRY
- *Spatial index* on column SOLID_GEOMETRY of table SURFACE_GEOMETRY
- *Normal index* on column GMLID of table CITYOBJECT
- *Normal index* on column GMLID of table SURFACE_GEOMETRY
- *Normal index* on column GMLID of table APPEARANCE
- *Normal index* on column GMLID of table SURFACE_DATA

The result of an index operation is reported in the console window. For instance, Figure 64 shows the result of a status query on both spatial and normal indexes. The status *ON* means that the corresponding index is enabled.

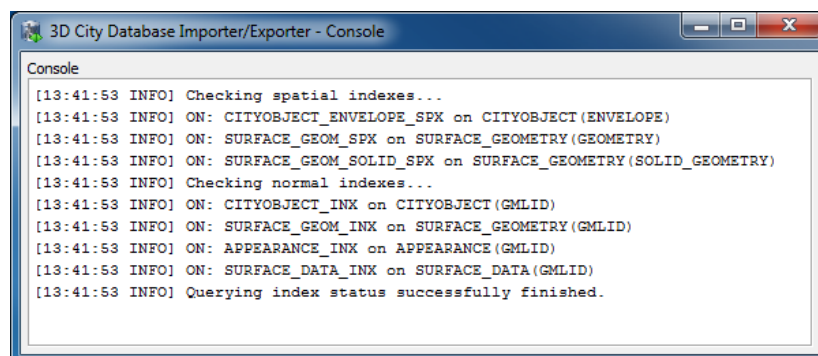


Figure 64: Result of a status query on spatial and normal indexes.

- Note:* It is *strongly recommended* to *deactivate the spatial indexes before running a CityGML import* on a *big amount of data* and to reactive the spatial indexes afterwards. This way the import will typically be a lot faster than with spatial indexes enabled. The situation may be different if only a small dataset is to be imported. Deactivating normal indexes should however never be required.
- Note:* Activating and deactivating indexes can take a long time, especially if the database fill level is high. Note that the operation **cannot be aborted** by the user since this could result in an inconsistent database state.

5.3 Importing CityGML files

In order to fill the 3D City Database instance with 3D city model content, the Importer/Exporter supports importing of CityGML files. Supported CityGML versions are 2.0.0, 1.0.0 and 0.4.0. The CityGML import operation is available on the `Import` tab of the operations window as shown below.

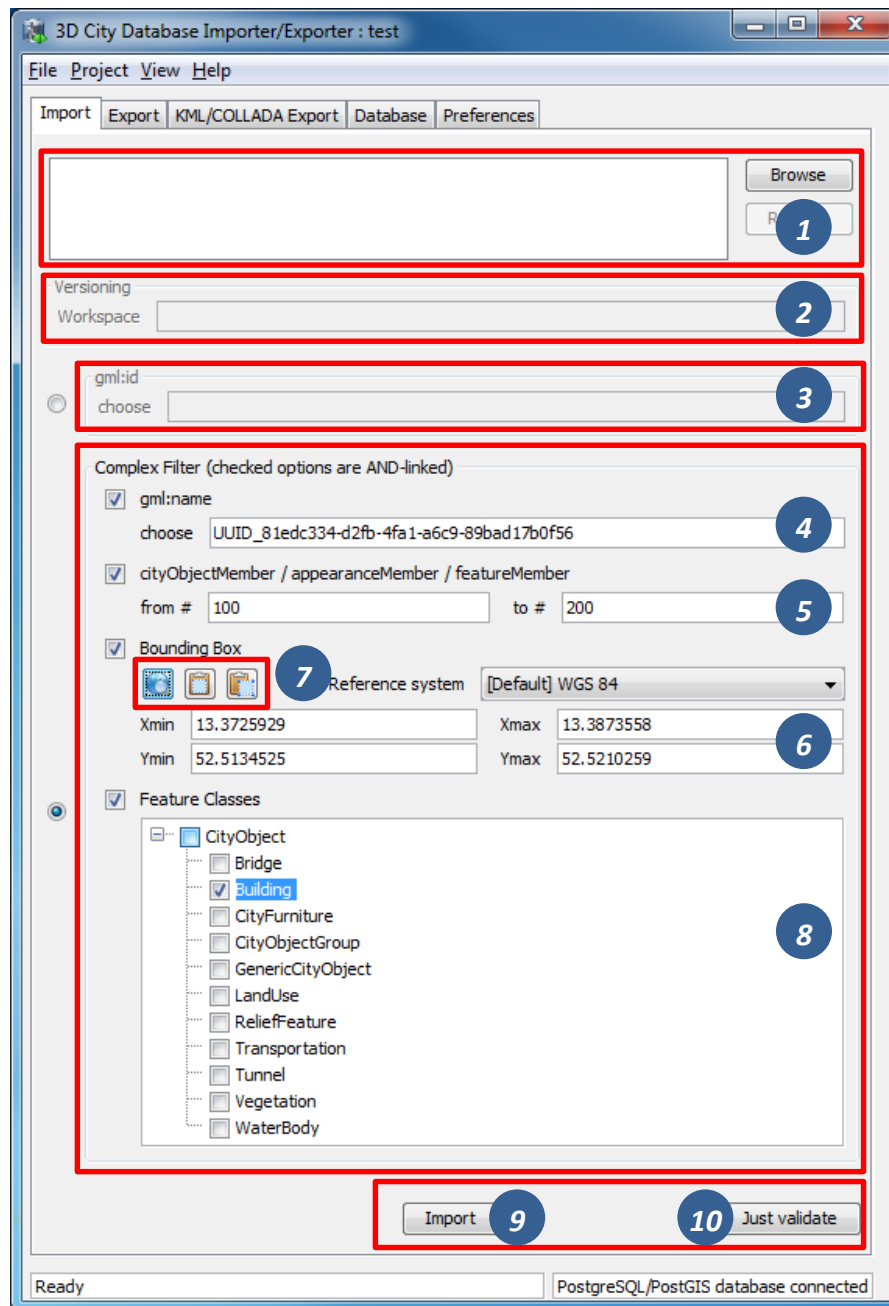


Figure 65: The CityGML import dialog.

Input file selection. At the top of the `Import` dialog [1], a list of one or more CityGML files to be imported has to be provided. Files can be selected through clicking on the *Browse* button, which opens a regular file selection dialog. Alternatively, you can drag&drop files from your preferred file explorer onto the `Import` tab. If the file list already contains entries, the drag&drop operation will replace its content. If you want to keep the previous entries and only

append additional files, keep the CTRL key pressed while dropping (on Windows). The *Remove* button or DEL key lets you remove selected entries from the input files. Note that adding folders to the list is also supported. Each folder will be recursively scanned for CityGML files, and every CityGML file found will be imported.

Workspace selection. If the 3D City Database instance is version-enabled (Oracle only), the name of the *workspace* into which the data shall be imported can be specified [2]. If no workspace is given, the default workspace is assumed (Oracle: *LIVE*).

Note: Importing into version-enabled tables typically takes *considerably more time* than importing into non-version-enabled tables. The import time can be reduced if spatial indexes are disabled beforehand.


Import filter. The import dialog allows for setting thematic and spatial filter criteria to narrow down the set of CityGML top-level features that are to be imported from the input files. Two radio buttons at the left side of the import dialog let you choose between a simple `gml:id` filter and a complex filter setting. For the `gml:id` filter, simply enter the `gml:id` that has to be matched by a CityGML feature [3]. More than one `gml:id` can be provided in a comma-separated list.

A complex filter setting may consist of a *gml:name filter*, a *counter filter*, a *spatial bounding box filter* and a *feature class filter*. A filter only applies if its checkbox is enabled. If more than one filter is chosen to be active, then the filter criteria are combined in a logical AND operation. If no checkbox is enabled, no filter criteria are applied and thus all CityGML features contained in the input file(s) will be imported.

- *gml:name filter* This filter takes a `gml:name` as parameter [4] and only imports CityGML features having a matching `gml:name`. Multiple `gml:name` values are not supported.
- *Counter filter* The counter filter lets you import a subset of the top-level features based on their position index over all input file(s). Simply provide the lower and upper boundary [5] for the position index to define the subset (both boundary limits are inclusive).
- *Bounding box filter* This filter takes a 2D bounding box as parameter that is given by the coordinate values of its lower left (*Xmin*, *Ymin*) and upper right corner (*Xmax*, *Ymax*) [6]. The bounding box is evaluated against the `gml:boundedBy` property of the CityGML input features.
- *Feature class filter* With the feature class filter, you can restrict the import to one or more CityGML features types by enabling the corresponding checkboxes [8]. Only features of the chosen type(s) will be imported.

Note: The filters only work on *top-level features* but *not on nested sub-features*.

For the *bounding box filter*, make sure that you choose a *coordinate reference system* from the drop-down choice list [6] that matches the provided coordinate values. Otherwise, the spatial filter may not work as expected. The coordinate reference system list can be augmented with user-defined reference systems (see chapter 5.6.4 for more information).

The coordinate values of the bounding box filter can either be entered manually or chosen interactively in a 2D map window. To open the map window, click on the map button  [7].

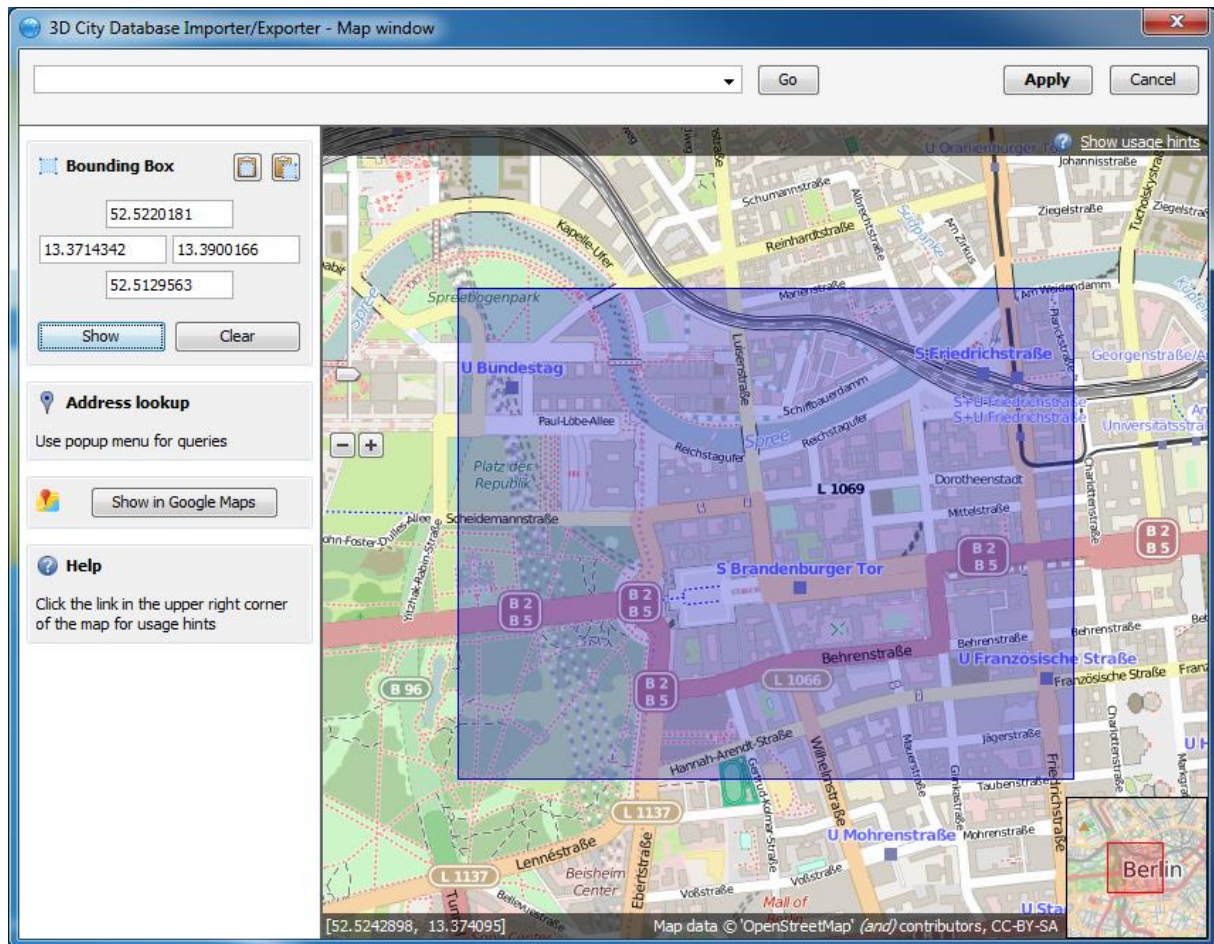




Figure 66: Bounding box selection using the 2D map window.

In the map window, keep the left mouse button clicked while holding the ALT key. This lets you draw a bounding box on the map. In order to move the map to a specific location or address, simply enter the location or address in the input field on top of the map and click the *Go* button, or use the map navigation controls. If you are happy with the bounding box selection, click the *Apply* button. This will close the map window and carry the coordinate values of the selected area into the corresponding fields of the bounding box filter [6]. Click *Cancel* if you want to close the map window but skip your selection. A more comprehensive guide on how to use the map window is provided in chapter 5.7.

With the  button on the bounding box filter dialog [7], you can copy a bounding box to the clipboard, while the  button pastes a bounding box from the clipboard to the input fields of the bounding box filter [6] (or use the right-click context menu).

XML validation. Before importing, the CityGML input files can be validated against the official CityGML XML schemas. Simply click the *Just Validate* button [10] in order to run the validation process. Filter settings are **not considered** in this process. Note that this operation does not require internet access since the XML schemas are packaged with the application. The CityGML features are **not imported** into the database during validation. The validation results are printed to the console window.

Note: It is **strongly recommended** that only CityGML files having successfully passed XML validation are imported into the database. Otherwise, errors in the data may lead to unexpected behavior or abnormal termination.

Import preferences. More fine-grained preference settings affecting the CityGML import are available on the *Preferences* tab of the operations window. Make sure to check these settings *before* starting the import process. A full documentation of the import preferences is available in chapter 5.6.1. The following table provides a brief summary overview.

Preference name	Description
<i>Continuation</i>	Metadata that is stored for every object in the database such as the data lineage, the updating person or the <code>creationDate</code> property.
<i>gml:id handling</i>	Generates UUIDs where <code>gml:ids</code> are missing on input features or replaces all <code>gml:ids</code> with UUIDs.
<i>Bounding box</i>	More settings on how to apply the bounding box filter.
<i>Address</i>	Controls the way in which xAL address fragments are imported into the database.
<i>Appearance</i>	Defines whether appearance information is imported.
<i>Geometry</i>	Allows for applying an affine transformation to the input geometry.
<i>Indexes</i>	Settings for automatically enabling/disabling spatial and normal indexes during imports.
<i>XML validation</i>	Performs XML validation automatically and exclude invalid features from being imported.
<i>Import log</i>	Creates a list of all successfully imported CityGML top-level features.
<i>Resources</i>	Allocation of computer resources used in the import operation.

Table 22: Summary overview of the import preferences.

CityGML import. Once all import settings are correct, the *Import* button [9] starts the import process. If a database connection has not been established manually beforehand, the currently selected entry on the *Database* tab is used to connect to the 3D City Database. The separate steps of the import process as well as all errors that might occur during the import are reported to the console window, whereas the overall progress is shown in a separate status window. The import process can be aborted at any time by pressing the *Cancel* button in the status window. The Importer/Exporter will make sure that all pending city objects are completely imported before it terminates the import process.

After having completed the import, a list enumerating the types and number of imported CityGML top-level features is printed to the console window as summary.

Note: The import operation does **not automatically apply a coordinate transformation** into the internal reference system of the 3D City Database instance. Thus, if the coordinate reference system of the CityGML input data does not match the coordinate

reference system defined for the 3D City Database instance, the user has to transform the coordinate values **before importing** the data (or use an affine transformation during import if this is sufficient). A possible workaround procedure can be realized as follows:

- 1) Set up a second (temporary) instance of the 3D City Database with an internal CRS matching the CRS of the CityGML instance document.
- 2) Import the dataset into this second 3D City Database instance.
- 3) Export the data from this second instance into the target CRS by applying a coordinate transformation (see CityGML export documentation in chapter 5.4).
- 4) The exported CityGML document now matches the CRS of the target 3D City Database instance and can be imported into that database. The temporary database instance can be dropped.

Note: The Importer/Exporter *does not check by any means* whether a *CityGML feature* from an input file *already exists* in the database. Thus, if an import is executed twice on the same dataset, all CityGML features contained in the dataset will be imported twice.

5.4 Exporting to CityGML

3D city model content stored in a 3D City Database instance can be fully or partially exported as CityGML models. The CityGML export is available on the **Export** tab of the operations window as depicted in the following figure.

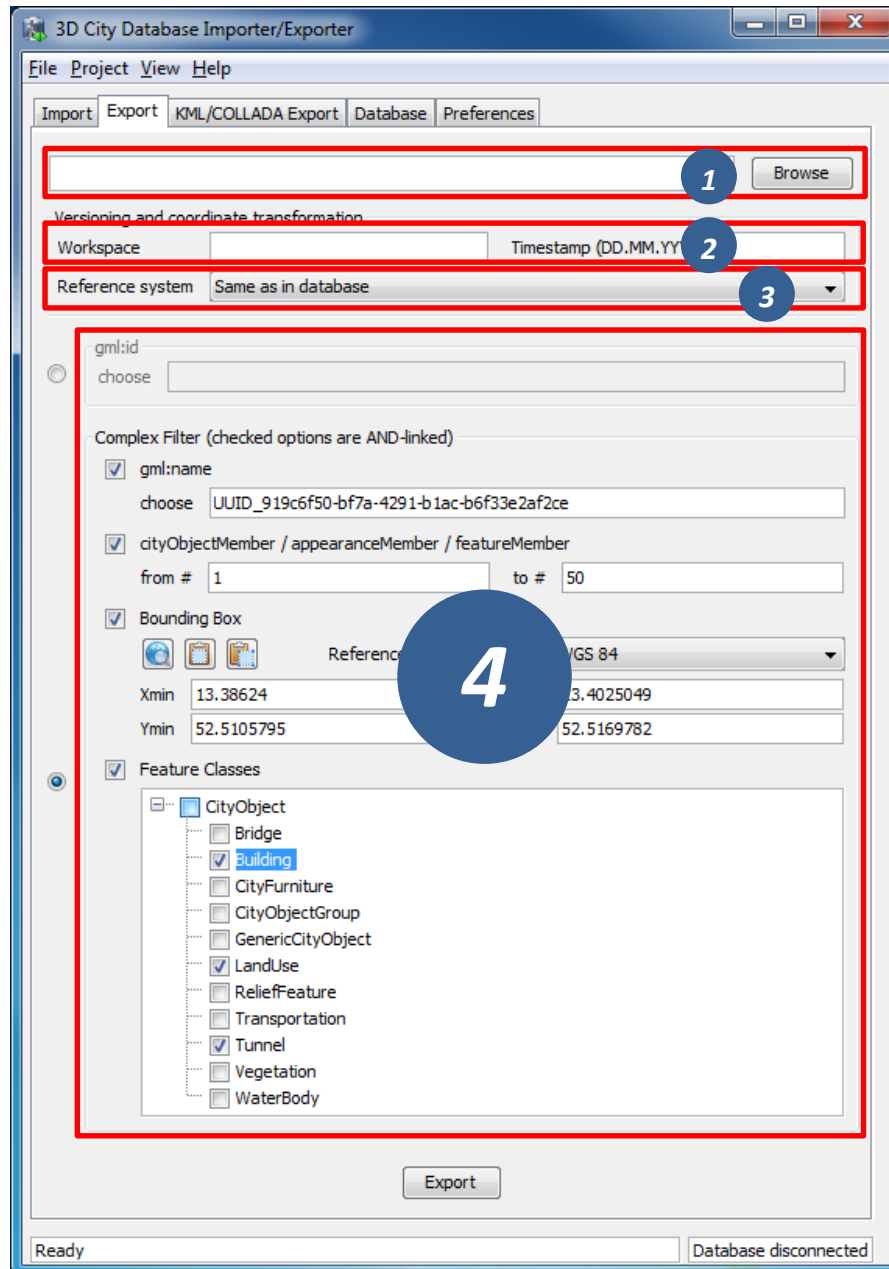


Figure 67: The CityGML export dialog.

Output file selection. At the top of the export dialog, the folder and filename of the target CityGML dataset have to be specified [1]. You can either manually enter the target file or open a file selection dialog via the *Browse* button. The export operation supports tiled exports, which typically results in multiple datasets being written to the file system. Nevertheless, also for tiled exports, only a single target file has to be specified. More details on tiled exports are provided in chapter 5.6.2.2.

Workspace selection. If the 3D City Database instance is version-enabled (Oracle only), the name of the *workspace* and the *timestamp* from which the data shall be exported can be specified [2]. If no workspace is provided, the default workspace is assumed (Oracle: *LIVE*).

Coordinate transformation. In general, coordinate values of geometry objects are associated with the coordinate reference system defined for the 3D City Database instance during setup, and are exported “as is” from the database. The export operation allows a user to apply a coordinate transformation into another reference system during export. The target coordinate reference system is chosen from the corresponding drop-down list [3]. This list can be augmented with user-defined reference systems (cf. chapter 5.6.4 for more details). When picking the entry “*Same as in database*”, no transformation will be applied (default behavior).

Export filter. Similar to importing CityGML datasets, the export operation supports thematic and spatial filter criteria in order to restrict exports to subsets of the 3D city model content. The corresponding filter dialog [4] is identical to the one on the `Import` tab. Please refer to chapter 5.3 for a description of the available filter settings.

Note: The `gml:name` filter internally uses an SQL `LIKE` operator and wildcards for identifying matches. For example, if you provide the string “*castle*” as parameter to this filter, this will be translated to “*LIKE '%castle%'*” in the query statement.

Note: When choosing a spatial *bounding filter*, make sure that *spatial indexes are enabled* so that filtering can be performed on the database (use the index operation on the Database tab to check the status of indexes, cf. chapter 5.2.2). Otherwise, the Importer/Exporter will do the filtering on the client side, which will run more slowly.

Note: The *feature class filter* in general behaves like for the CityGML import. However, regarding *city object groups* the following rules apply:

- 1) If only the feature type *CityObjectGroup* is checked, then all city object groups and all their group members (independent of their feature type) are exported.
- 2) If further feature types are selected in addition to *CityObjectGroup*, then only group members matching those feature types are exported. Of course, all features that match the type selection but are not group members are also exported.

Export preferences. In addition to the settings on the `Export` tab, more fine-grained preference settings affecting the CityGML export are available on the `Preferences` tab of the operations window. Make sure to check these settings before starting the export process. A full documentation of the export preferences is available in chapter 5.6.2. The following table provides a brief summary overview.

Preference name	Description
<i>CityGML version</i>	CityGML version to be used for exports.
<i>Bounding box</i>	More settings on how to apply the bounding box filter. Most importantly, tiled exports are configured here.
<i>CityObjectGroup</i>	Defines whether group members are exported by value or by reference.
<i>Address</i>	Controls the way in which xAL address fragments are exported from the database.

<i>Appearance</i>	Defines whether appearance information is exported.
<i>XLinks</i>	Controls whether referenced features or geometry objects are exported using XLinks or as deep copies.
<i>Resources</i>	Allocation of computer resources used in the export operation.

Table 23: Summery overview of the export preferences.

CityGML export. Having completed all settings, the CityGML data export is triggered with the *Export* button at the bottom of the dialog (cf. Figure 67). If a database connection has not been established manually beforehand, the currently selected entry on the *Database* tab is used to connect to the 3D City Database. Progress information is displayed within a separate status window. This status window also offers a *Cancel* button that lets a user abort the export process. The separate steps of the export process as well as possible error messages are reported to the console window.

5.5 Exporting to KML/COLLADA

3D City Database contents can be directly exported in KML [Wilson 2008] and COLLADA [Barners & Finch 2008] formats for presentation, viewing, and visual inspection in a broad range of applications such as earth browsers like Google Earth, ArcGIS and ArcGIS Explorer. Built-in support for object highlighting and generic creation of KML information balloons facilitate the interactive exploration of your 3D city models.

Note: KML/COLLADA formatted exports come straight from the 3D City Database. No direct file transformation CityGML → KML/COLLADA is supported yet. If a CityGML file shall be converted to KML/COLLADA, the CityGML content must be imported into the database first and then exported into the KML/COLLADA format.

The *KML/COLLADA Export* tab shown in Figure 68 collects all parameters required for the export in a similar fashion as for a CityGML export (see the previous chapter). In addition, more fine-grained preference settings affecting the KML/COLLADA export are available on the *Preferences* tab of the operations window. Make sure to check these settings before starting the export process. A full documentation of the export preferences is available in chapter 5.6.3. The following table provides a brief summary overview.

Preference name	Description
<i>General Preference</i>	Some common settings of the exported files
<i>Rendering Preferences</i>	Defines the look of the KML/COLLADA exports when visualized in the virtual globes (e.g. Cesium, Google Earth, NASA World Wind, ESRI ArcGlobe). Each of the top-level feature categories has its own Rendering settings here
<i>Information Balloon Preferences</i>	KML offers the possibility of enriching its placemark elements with information bubbles, so-called balloons. They can be specified here
<i>Altitude/Terrain Preference</i>	Controls the way through which the exported datasets to be perfectly displayed in the Earth browser

Table 24: Summery overview of the KML/COLLADA export preferences.

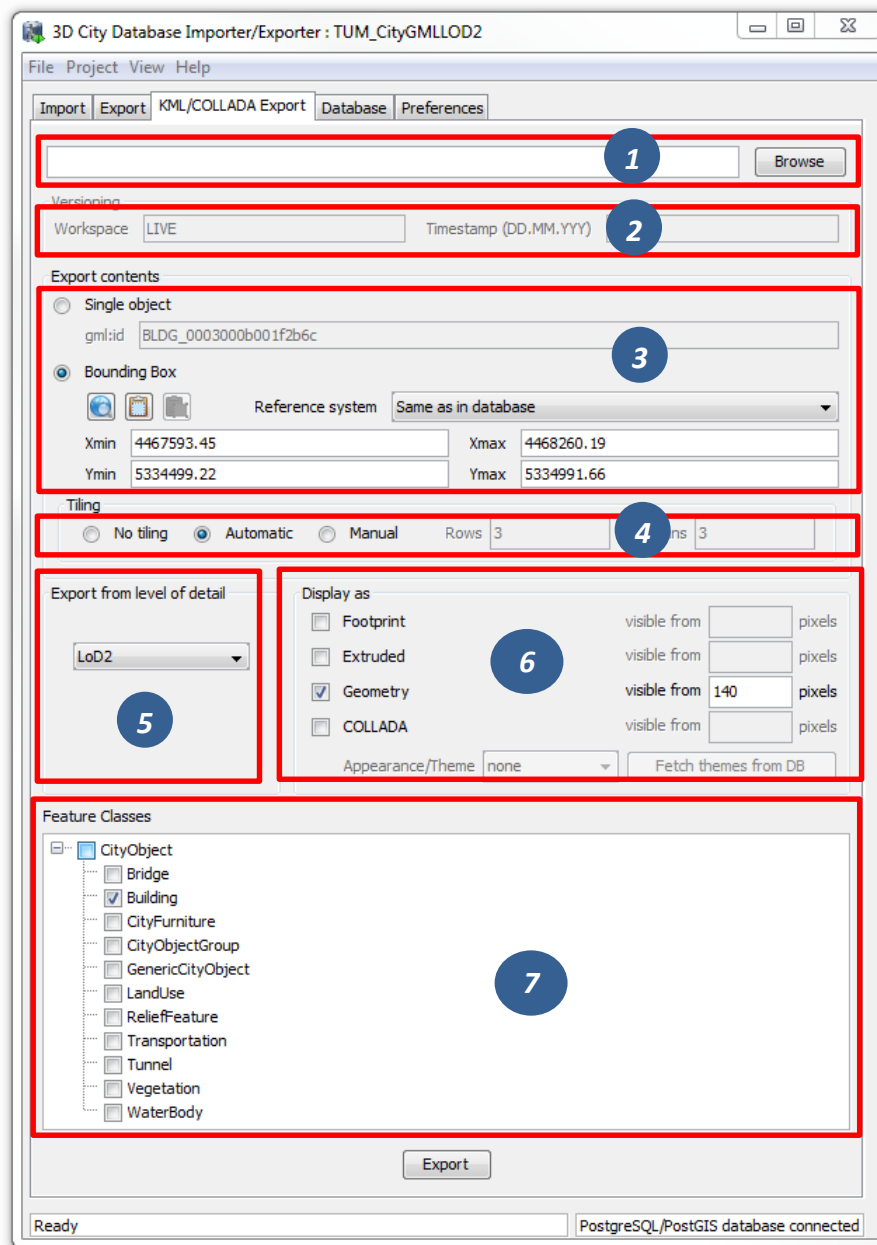


Figure 68: The KML/COLLADA Export tab allowing for exporting KML/COLLADA models from the 3DCityDB.

Output file selection. Type the filename directly into the text field or activate the file dialog provided by the operating system after pushing the *Browse* button [1].

Workspace selection. If the 3D City Database instance is version-enabled (Oracle only), the name of the *workspace* and the *timestamp* from which the data shall be exported can be specified [2]. If no workspace is provided, the default workspace is assumed (Oracle: *LIVE*).

Export contents. These KML/COLLADA Exporter allows for specifying/selecting the objects of interest for the export. These can be single objects or whole areas delimited by a bounding box. Two radio buttons [3] at the left side of the export dialog let you choose between those two options.

- *Single object*: Enter the GML IDs of the object(s) of interest. Multiple IDs have to be separated by commas.
- *Bounding Box*: Enter the coordinates of a bounding box defining the area of interest. Objects are exported if they are fully covered by the specified bounding box or if they intersect with its left or bottom borders. This strategy also applies to tiled exports (objects in a tile are only exported when fully covered by the tile's bounding box or intersecting with the tile's left or bottom borders). This is done in order to avoid exporting the same object twice when the object lies at the same time on more than one tile. The reference system used for defining the bounding box can be the same as the one used in the database or any other one supported by Oracle and PostGIS. It is also possible to add further user-defined reference systems was introduced (see the previous chapter). New reference systems can be added to the Import/Export tool (preferences tab, node *Database*, subnode *Reference systems*) as long as they are supported by the used database server. The target system with the same dimensionality (WGS84 for 2D, WGS84 3D for 3D) will be applied for the coordinate transformation during the KML/COLLADA Export.

Tiling only applies to exports of areas defined by a bounding box. Tiled exports are used in order to load and unload parts of the exported model depending on their current visibility when viewed, for example, in Google Earth. Since the Earth Browser's responsiveness decreases greatly with single files larger than 10 Mb, tiled exports (with tile file sizes usually a lot smaller than that) are highly recommended. As mentioned above, only objects fully covered by the tile's bounding box or intersecting with the tile's left or bottom borders will be exported.

There are three tiling modes [4] available for a KML/COLLADA export:

- *no tiling*: as the name implies, no tiling takes place. Just a single file is exported. This is only advisable when the resulting file is at most 10 Mb in size.
- *automatic*: the area enclosed by the bounding box will be exported in tiles having roughly the side length set on the preferences tab under the node *KML/COLLADA Export*, subnode *Rendering* (default value is 125m.). The amount of exported rows and columns will be calculated by dividing the length and width (in unit of meters) of the delimiting bounding box by the preferred tile side length and rounding up the result. For example: if the user wants to export a 1000m x 1100m bounding box with a preferred tile side length of 300m, 4x4 tiles will be generated since $1000/300 = 3.333$ and $1100/300 = 3.666$. This also implies: in case of automatic tiling it cannot be guaranteed that tiles will be perfectly square, but they will tend to.
- *manual*: the number of rows and columns can be freely set by the user. The area will be divided in equally spaced portions horizontally and vertically and the resulting tile sizes and forms will adapt to the values specified.

All these tiling modes can be combined with the (Preferences → General) option “Each CityObject in an own region” (cf. chapter 5.6.3.1 for more details), which for each object exported defines a region limited by its envelope coordinates. The single object regions may have different visibility settings to those of the tiles containing them, so it can be possible that

while a tile as a whole is visible some objects inside are not because their own region visibility conditions are not yet matched.

Untiled exports (no tiling) and tiled exports of type automatic or manual will contain one main kml file, so-called master file, pointing to all export contents. In case of tiled exports each tile filename will be enhanced with the tile's row and column number as a suffix.

Export from level of detail. The Level of Detail as defined by the CityGML specification should be used as basis information for the KML/COLLADA export. For the same city object higher levels of detail usually contain many more geometries and these geometries are more complex than in lower levels. For instance, a building made of 40 polygons in LoD2 may consist of 3000 polygons in LoD3. This means LoD3 based exports are a lot more detailed than LoD2 based exports, but they also take longer to generate, are bigger in size and therefore load more slowly in the Earth browser.

By using the drop-down list [5] a single constant LoD can be used as basis for all exports or it can be left to the importer/exporter tool to automatically determine which the highest LoD available for each cityobject is and then use it as the basis for the KML/COLLADA exports.

Display as. These fields in the export dialog [6] determines what will be shown when visualizing the exported dataset in Google Earth.

- *Footprint:* objects are represented by their ground surface projected onto the earth surface. This is a pure KML export.
- *Extruded:* objects are represented as blocks models by extruding their footprint to their measured height (thematic CityGML attribute), which must be filled with a proper value in m. Pure KML export.
- *Geometry:* shows the detailed geometry of ground, wall, and roof surfaces of buildings and appearance information. It shows the different thematic surfaces by means of coloring them (textures are not supported by KML) according to the settings in the preferences tab, *KML/COLLADA Export* node, *Rendering* subnode. If not explicitly modeled, thematic surfaces will be inferred for LoD1 or LoD2 based exports following a trivial logic (surfaces touching the ground –that is, having a lowest z-coordinate- will be considered wall surfaces, all other will be considered roof surfaces), in LoD3 or LoD4 based exports surfaces not thematically modeled will be colored as wall surfaces.
- *COLLADA:* shows the detailed geometry including support for textures. The Appearance/Theme combo box below allows choosing from all possible appearance themes (as defined in the CityGML specification [Gröger et al. 2012]) available in the currently connected 3DCityDB. The list is workspace- and timestamp sensitive and will be filled on demand when clicking on the *fetch* button. Default value is *none*, which renders no textures at all and colors all surfaces in a neutral gray tone. Export consists of KML and COLLADA parts.

Note: For Oracle, the *Footprint* and *Extruded* display forms internally use the spatial function `SDO_AGGR_UNION`. This function is not allowed to be used under Oracle 10g/11g with the *Locator* license option even if it happens to be available. The Importer/Exporter does not check the Oracle license option. Thus, it is up to the user to observe the Oracle

license and not to use the *Footprint* and *Extruded* display forms under Oracle 10g/11g *Locator*. This restriction does not hold for the Oracle *Spatial* license option. Likewise, starting from Oracle 12c, `SDO_AGGR_UNION` is also available for *Locator*.

Depending on the level of detail chosen as basis some display form checkboxes will become enabled or disabled, depending on whether the level of detail offers enough information for this display form or not. Footprint can be exported from any CityGML LoD (0 to 4), whereas Extruded, Geometry, and COLLADA exports are possible from LoD1 upwards. Exports will have their filename enhanced with a suffix specifying the selected display form. This applies for both tiled and untiled exports.



Figure 69: The same building displayed as (top down and left to right) footprint, extruded, geometry, COLLADA.

With the visibility field next to each display form the user can control the KML element `<minLodPixels>`, see [Wilson 2008]: measurement in screen pixels that represents the minimum limit of the visibility range for a given `<Region>`. A `<Region>` is in the generated tiled exports equivalent to a tile. The `<maxLodPixels>` value is identical to the `<minLodPixels>` of the next visible display form, so that display forms are seamlessly switched when the viewer zooms in or out. The last visible display form has a

<maxLodPixels> value of -1, that is, visible to infinite size. Visibility ranges can start at a value of 0 (they do not have to, though). Please note that the region size in pixels depends on the chosen tile size. Thus, if the tile size is reduced also the visibility ranges should be reduced. Increases in steps of a third of the tile side length are recommended. An example of a good combination for a tile size of about 250m x 250m could be: *Footprint*, visible from 50 pixels, *Geometry*, visible from 125 pixels, *COLLADA*, visible from 200 pixels. Some display forms, like *Extruded* in this example, can be skipped. The visibility field only becomes enabled for bounding box exports; single building exports are always visible.

Feature Classes. Similar to CityGML imports and exports it is also possible to select what top-level feature types shall be displayed in a KML/COLLADA export.



Figure 70: Example for exported CityGML top-level features (building, bridge, tunnel, water, vegetation, transportation etc.) displayed as KML/COLLADA

With the selection tree panel [7] it is possible to pick each category individually and also leave single categories out, i.e.: export *CityFurniture* and *WaterBody* only, or export everything but *Building* and so on. Between LoD1 and LoD4 all feature types are available. For LoD0 only those top-level feature types offering LoD0 geometry in the CityGML 2.0 schema (*Building*, *Waterbody*, *LandUse*, *Transportation* and *GenericCityObject*) are selectable, whereas the rest of the feature class checkboxes will become automatically disabled.

Note: Support for *Relief* features in KML/COLLADA exports is currently limited to the type *TIN_RELIEF*. Other *Relief* types such as *MASSPOINT_RELIEF*, *BREAKLINE_RELIEF*, and *RASTER_RELIEF* are not supported currently. Also, due to the usually wide-stretched area of *Relief* features and the non-clipping nature of the BoundingBox filter it is recommended to export *Relief* features in a single step making use of the *no tiling* option and using an extensive enough BoundingBox. As an alternative, the digital terrain model data can be divided in smaller *ReliefComponents* tailored to match the tiling settings of the desired export (their area

contained in or equal to the resulting tiles). This requires altering the original data nevertheless and, as such, it must be done before the CityGML contents are imported into the database at all.

KML/COLLADA export. Having completed all settings, the KML/COLLADA data export is triggered with the *Export* button at the bottom of the dialog (cf. Figure 68). If a database connection has not been established manually beforehand, the currently selected entry on the Database tab is used to connect to the 3D City Database. Progress information is displayed within a separate status window. This status window also offers a *Cancel* button that lets a user abort the export process. The separate steps of the export process as well as possible error messages are reported to the console window.

5.5.1 Support for Point and Line Geometries of GenericCityObject

In the current release of KML/COLLADA exporter only surface-based and volumetric representations of the CityGML top-level feature types are supported except *GenericCityObject*. The feature type *GenericCityObject* as defined within CityGML's Generics module (see chapter 2.2.4.6) allows for modeling and exchanging of 3D city objects which are not covered by any other thematic module of the CityGML data model. In every LOD, the spatial characteristics of a *GenericCityObject* can be described by an arbitrary 3D geometry object being a subtype of *gml:_Geometry* (the root of the GML geometry hierarchy). Thus, a *GenericCityObject* may be geometrically represented by one or more points, lines, surfaces, or volume objects per LOD. As for example, *GenericCityObjects* can be used to model network structures such as indoor routing networks (which are not predefined by the current CityGML specification) with every *GenericCityObject* representing either a junction or an edge of the network.

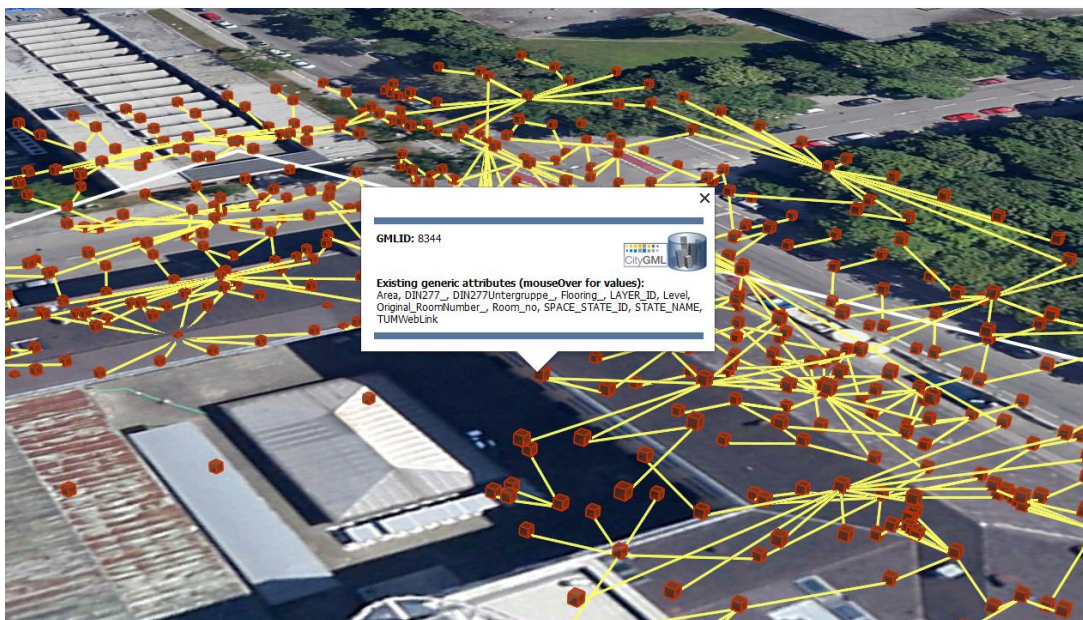


Figure 71: Visualization of the network model of the building interior of Technical University Munich (TUM)

According to the CityGML specification, a *GenericCityObjects* feature may have at most one *gml:_Geometry* representation in each LOD. Correspondingly, per LOD, a *GenericCityObject* may either has a surface-based geometry or a single point respectively line geometry but not

both. When exporting a *GenricCityObject*, first the corresponding database table is scanned whether it holds a geometry representation of the feature for a given LOD. If no surface-based geometry is found, point or line geometry will be continually searched and exported if found.

Note: The exported point and line geometries are not contained in separated kml files, but attached to the output files whose filename is enhanced with a suffix specifying the selected display form e.g. *Footprint*, *Extruded*, *Geometry*, and *COLLADA*.

5.5.2 Loading exported models in Google Earth

Usage of the most up-to-date Google Earth version is highly recommended (starting from January 2015 Google Earth Pro can be used free of charge). Some of the features described in this documentation, like highlighting, require version 6.0.1 or higher to work flawlessly (they do work in older versions, but not as smooth).

Displaying a file in Google Earth can be achieved by opening it through the menu ("*File*", "*Open*") or double-clicking on any kml or kmz file if these extension are associated with the program (default option at Google Earth's installation time).

Loaded files can be refreshed when generated again after loading (if for example the balloon template file was changed) by choosing the "*Revert*" option in the context menu on the sidebar. There is no need to delete and load them again or shutdown or restart the Earth browser.

For best performance, cache options ("*Tools*", "*Options*", "*Cache*") should be set to their maximum values, 1024MB for memory cache size, 2000MB for disk cache. Actual maximums may be lower depending on the computer's hardware.

Show Terrain ("*Tools*", "*Options*", "*3D View*") should be enabled, quality set at around 1/3. Disable it only in the case exports cannot be seen although shown as loaded in Google Earth's sidebar: they are probably buried into the ground (see chapter 5.6.3.4), and remember to enable showing of the terrain again when loading the next unrelated exports.

When exporting balloons into individual files (one for each object) written together into a *balloon* directory access to local files and personal data must be allowed ("*Tools*", "*Options*", "*General*"). Google Earth will issue a security warning that must be accepted, otherwise the contents of the balloons (when in individual files and not as a part of the doc.kml file) will not be displayed.

It is also possible to upload the generated KML/COLLADA files to a web server and access them from there as a *Network Link* in Google Earth or via internet browser with installed Google Earth Plugin. In this case, as stated in chapter 5.6.3.1, export in kmz format is recommended and balloon content must be part of the doc.kml file. Due to security risks, the Plugin has no option for allowing access to local files and personal data.

Note: Starting with version 7 (and at least up to version 7.1.1.1888) Google Earth has changed the way transparent or semi-transparent surfaces are rendered. This is especially relevant for visualizations containing highlighting surfaces (explained in chapter 5.6.3.2). When viewing KML/COLLADA export results in Google Earth it is strongly recommended to use Google Earth version 7 or higher and switch to the OpenGL graphic mode for an optimal viewing experience. Changing the Graphic Mode can be achieved by clicking on *Tools, Options* entry, *3D View* Tab.

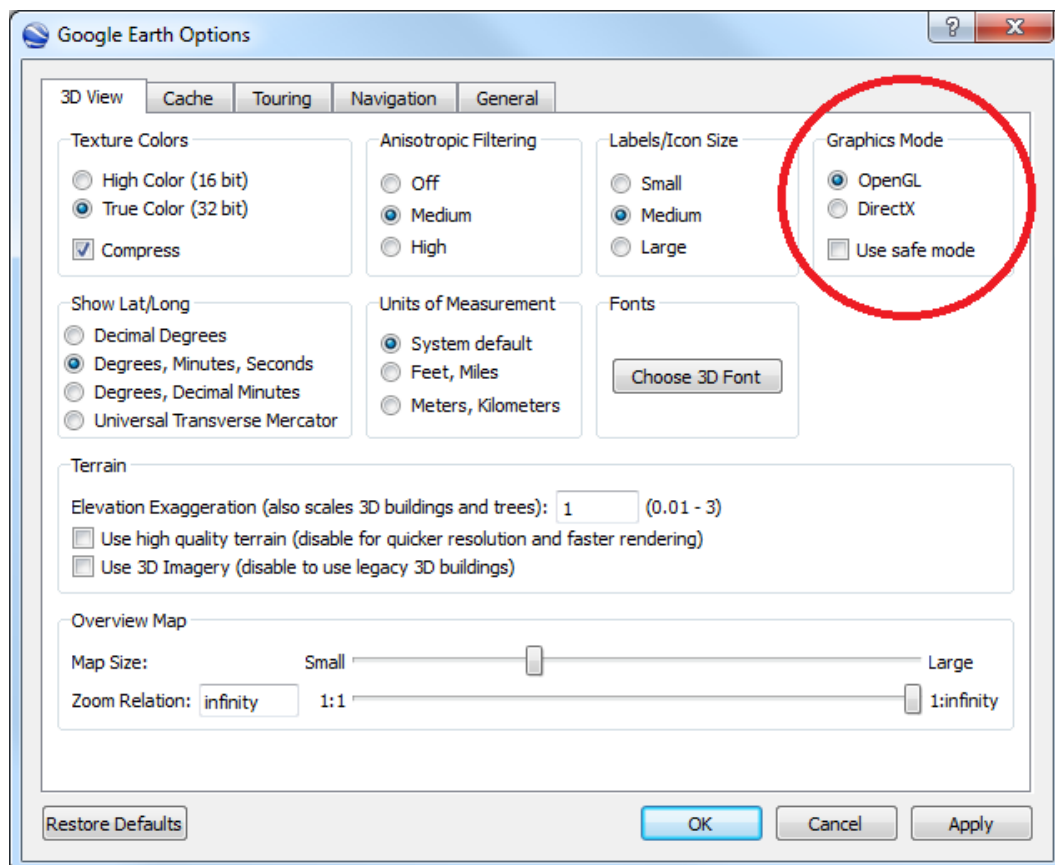


Figure 72: Setting the Graphics Mode in Google Earth



Figure 73: KML/COLLADA export rendered with DirectX, highlighting surface borders are noticeable everywhere

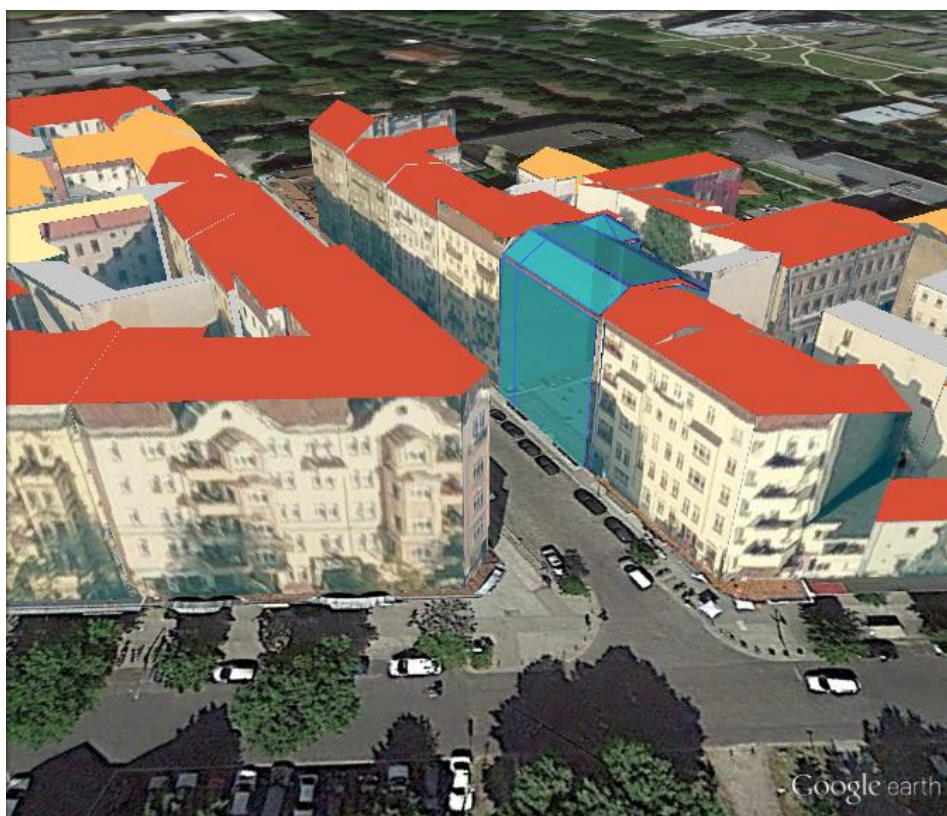


Figure 74: The same scene rendered in OpenGL mode

5.6 Preferences

In addition to the settings on the Import, Export, KML/COLLADA Export and Database tabs of the operations window, more preferences affecting the separate operations of the Importer/Exporter are available on the Preferences tab shown below.

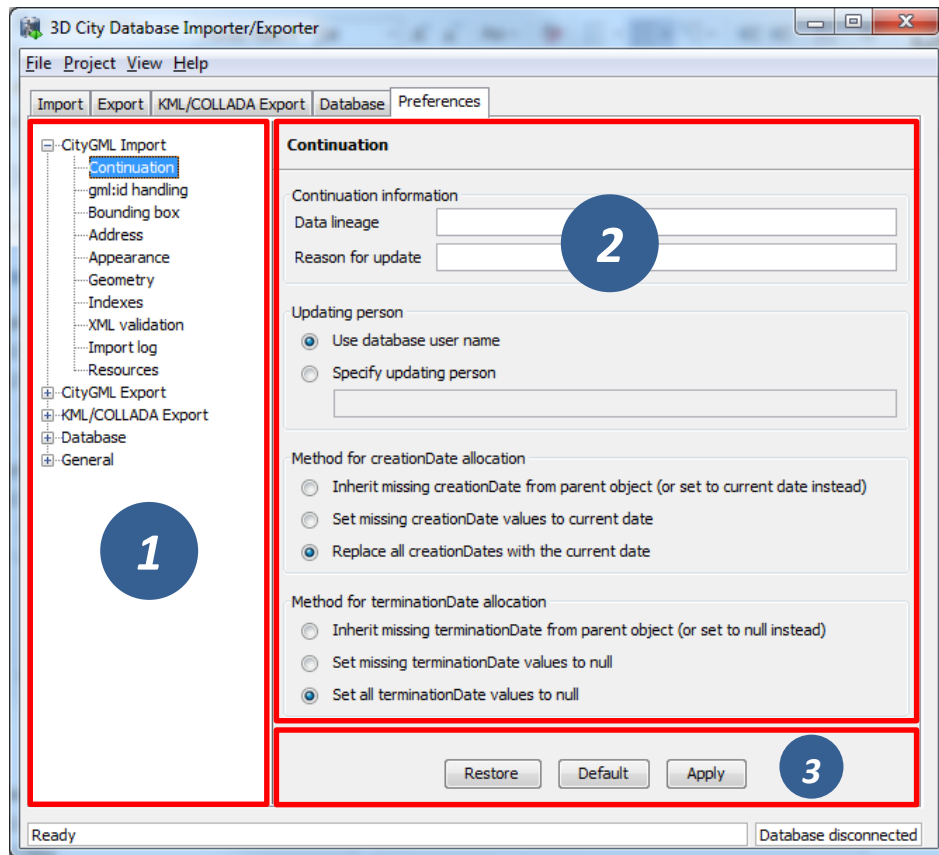


Figure 75: The preferences dialog.

The preferences are structured in a tree view [1] on the left side of the dialog with the following main nodes:

- CityGML Import Settings affecting the CityGML import operation
- CityGML Export Settings affecting the CityGML export operation
- KML/COLLADA Export Settings affecting the KML/COLLADA export operation
- Database Database-specific settings
- General General settings affecting the entire application

Below these main nodes, further subnodes organize the preferences into separate topics. When selecting a node in the tree view, the associated settings dialog is displayed on the right side [2]. Changes made to the settings of the selected node are applied through the *Apply* button [3]. The buttons *Restore* and *Default* allow for resetting the preferences to their previous state or to their default values.

The preferences (including the settings on the separate operation tabs) are automatically stored in the config file of the Importer/Exporter and are restored from this file upon program start.

Thus, changes made to the preferences are remembered on restart. Via the `Project` menu available from the menu bar of the Importer/Exporter, the preferences can optionally be stored in or loaded from user-defined config files (cf. chapter 5.1).

5.6.1 CityGML import preferences

5.6.1.1 Continuation

The `Continuation` preferences allow for specifying metadata that is assigned to every city object at import time. The metadata is carried to columns of the table `CITYOBJECT` and is therefore accessible in SQL queries.

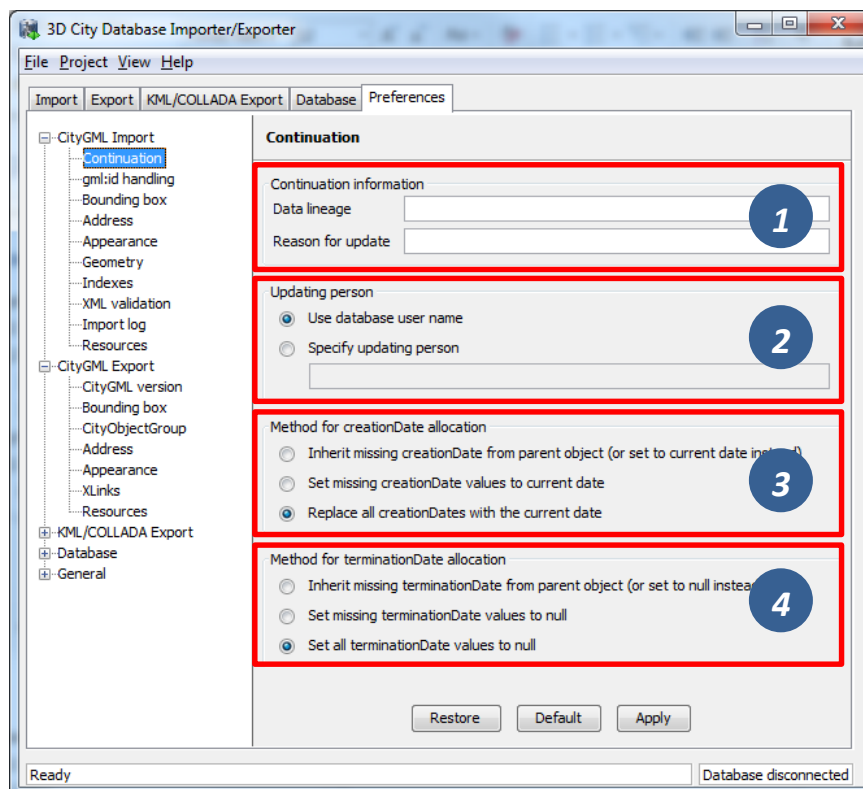


Figure 76: CityGML import preferences – Continuation.

The following metadata can be set:

Metadata	Description
<i>Data lineage</i> [1]	A string value denoting the origin of the data. (column: <code>LINEAGE</code> ; default value: <code>NULL</code>)
<i>Reason for update</i> [1]	A string value providing the reason for a data update. (column: <code>REASON_FOR_UPDATE</code> ; default value: <code>NULL</code>)
<i>Updating person</i> [2]	A string value identifying the person being responsible for importing or updating the city object. (column: <code>UPDATING_PERSON</code> ; default value: name of the database user)
<i>creationDate</i> [3]	A timestamp value denoting the date of creation of the city object. If this date is not available from the CityGML feature during import, it may either be set to the import date or be inherited from the parent feature (if available). Alternatively, the user can choose to replace all creation dates from the input files with the import date. (column: <code>CREATION_DATE</code> ; default value: import date)

<i>terminationDate</i> [4]	A timestamp value denoting the date of termination of the city object. If this date is not available from the CityGML feature during import, it may either be set to NULL or be inherited from the parent feature (if available). Alternatively, the user can choose to replace all termination dates in the input files with NULL. (column: <code>TERMINATION_DATE</code> ; default value: NULL)
----------------------------	--

Table 25: Metadata stored with every city object in the table CITYOBJECT.

Note: Both *creationDate* and *terminationDate* are CityGML properties of city objects and therefore are exported to CityGML datasets. The remaining metadata information does not map to CityGML properties. It is therefore not exported to CityGML datasets but is only available in the database.

5.6.1.2 gml:id handling

Globally unique object identifiers are crucial for ensuring data consistency and for enabling data management workflows. Especially when it comes to (subsequently) updating the city model content in the database, unique identifiers will help to quickly identify and replace objects in the database with candidates from external datasets. Unfortunately, `gml:id` values do not meet the requirement of global uniqueness since they are, per definition, optional and only unique within the scope of a single dataset.

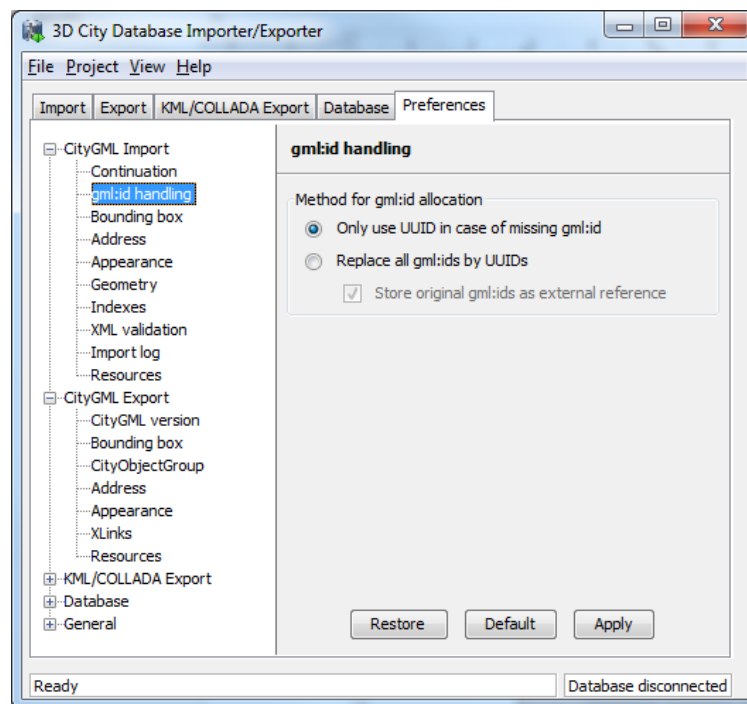


Figure 77: CityGML import preferences – `gml:id` handling.

Per default, the Importer/Exporter assumes that the `gml:id` values associated with the city objects to be imported are globally unique and therefore imports them “as is” into the database. Only in case a city object (or geometry object) lacks a `gml:id`, a UUID value will be generated at import time and stored with the object.

This default behavior can be overridden with this preferences dialog in order to let the Importer/Exporter replace all `gml:id` values in the input file(s) with generated UUID values.

Use this option with caution. The original `gml:id` value may optionally be stored as external reference to not lose this information.

5.6.1.3 Bounding box

On the Import tab, a *bounding box filter* can be defined in order to only import features within a given geographic area (cf. chapter 5.3). The provided bounding box is evaluated against the `gml:boundedBy` property of the input features. Per default, features whose bounding box is inside or overlaps with the filter geometry are imported.

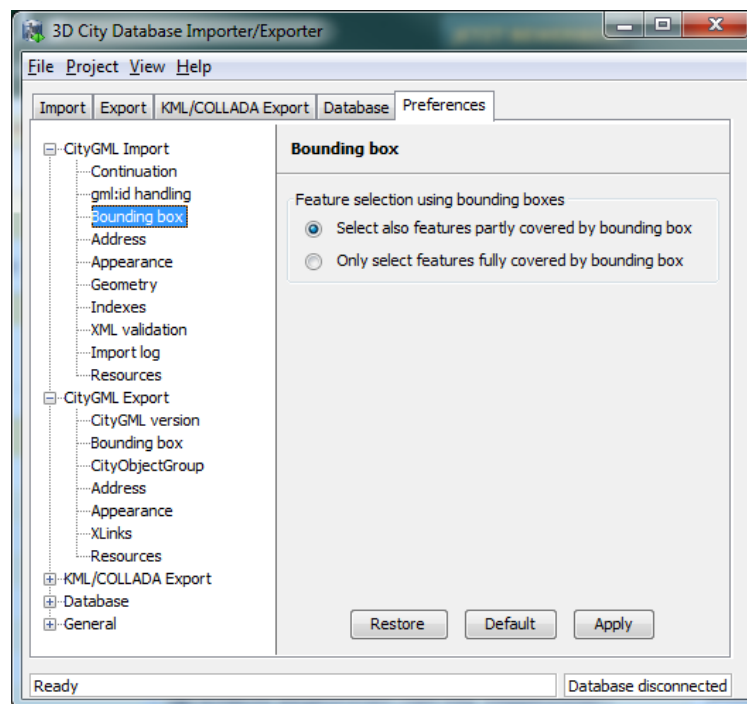


Figure 78: CityGML import preferences – Bounding box.

This preference dialog lets the user change the behavior of the bounding box filter to be more restrictive in that only those features whose bounding box is completely inside the filter geometry are imported.

5.6.1.4 Address

CityGML relies upon the *OASIS Extensible Address Language* (xAL) standard for the representation and exchange of address information. xAL provides a flexible and generic framework for encoding address data according to arbitrary address schemes. The columns of the ADDRESS table of the 3D City Database however only map the most common fields in address records (cf. chapter 2.3). Moreover, the Importer/Exporter currently does not support arbitrary xAL fragments but is tailored to the parsing of following two xAL templates that are taken from the CityGML specification.

```
<bldg:Building>
...
  <bldg:address>
    <Address>
      <xalAddress>
        <!-- Bussardweg 7, 76356 Weingarten, Germany -->
        <xAL:AddressDetails>
          <xAL:Country>
```



```

    <xAL:CountryName>Germany</xAL:CountryName>
    <xAL:Locality Type="City">
      <xAL:LocalityName>Weingarten</xAL:LocalityName>
      <xAL:Thoroughfare Type="Street">
        <xAL:ThoroughfareNumber>7</xAL:ThoroughfareNumber>
        <xAL:ThoroughfareName>Bussardweg</xAL:ThoroughfareName>
      </xAL:Thoroughfare>
      <xAL:PostalCode>
        <xAL:PostalCodeNumber>76356</xAL:PostalCodeNumber>
      </xAL:PostalCode>
    </xAL:Locality>
  </xAL:Country>
</xAL:AddressDetails>
</xalAddress>
</Address>
</bldg:address>
</bldg:Building>

```

```

<bldg:Building>
...
  <bldg:address>
    <Address>
      <xalAddress>
        <!-- 46 Brynmaer Road Battersea LONDON, SW11 4EW United Kingdom -->
        <xAL:AddressDetails>
          <xAL:Country>
            <xAL:CountryName>United Kingdom</xAL:CountryName>
            <xAL:Locality Type="City">
              <xAL:LocalityName>LONDON</xAL:LocalityName>
              <xAL:DependentLocality Type="District">
                <xAL:DependentLocalityName>Battersea
                </xAL:DependentLocalityName>
              <xAL:Thoroughfare>
                <xAL:ThoroughfareNumber>46</xAL:ThoroughfareNumber>
                <xAL:ThoroughfareName>Brynmaer Road
                </xAL:ThoroughfareName>
              </xAL:Thoroughfare>
            </xAL:DependentLocality>
          <xAL:PostalCode>
            <xAL:PostalCodeNumber>SW11 4EW</xAL:PostalCodeNumber>
          </xAL:PostalCode>
        </xAL:Locality>
      </xAL:Country>
    </xAL:AddressDetails>
  </xalAddress>
</Address>
</bldg:address>
</bldg:Building>

```

Figure 79: xAL fragments supported by the Importer/Exporter.

If xAL address information in a CityGML instance document does not comply with one of the templates (e.g., because of additional or completely different entries), the address information will only partially be stored in the database (if at all). In order to not lose any original address information, the entire `<xal:AddressDetail>` XML fragment can be imported “as is” from the input CityGML file and stored in the `XAL_SOURCE` column of the `ADDRESS` table in the 3D City Database.

For this purpose, simply check the *Import original <xal:AddressDetail> XML* option (this is the default value). Note that the import of the XML fragment does not affect the filling of the remaining columns of the `ADDRESS` table (`STREET`, `HOUSE_NUMBER`, etc.) from the xAL address information.

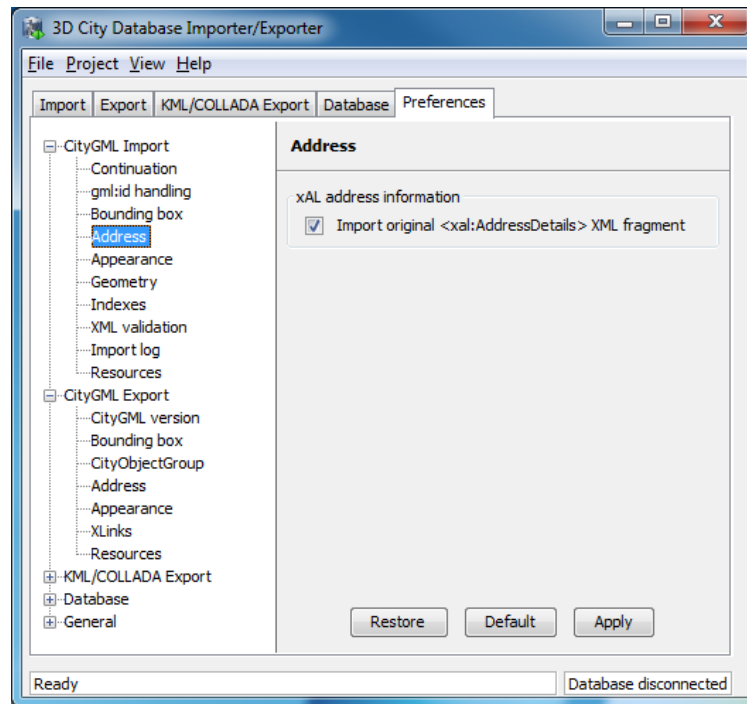


Figure 80: CityGML import preferences – Address.

The symmetrical setting for CityGML exports (i.e., recovering the xAL fragment from XAL_SOURCE) is explained in chapter 5.6.2.4.

5.6.1.5 Appearance

The Appearance preference settings define how appearance information (i.e., materials and textures associated with the observable surfaces of a city object) is processed at import time.

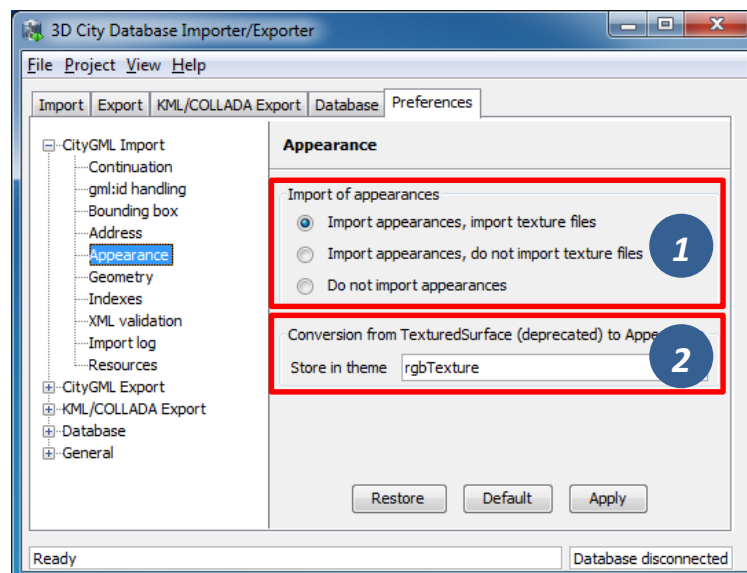


Figure 81: CityGML import preferences – Appearance.

Per default, all appearance information as well as all related texture image files are loaded into the 3D City Database [1]. The Importer/Exporter will work on both image files located in a relative path to the CityGML dataset and image files referenced by a valid URL. The latter might require network access. Alternatively, a user may choose to only consider the appearance

information but to not load the texture image files. As a third option, appearance information can be completely skipped during import [1].

Prior to version 1.0 of the CityGML standard, material and texture information of surface objects was modelled using the `TexturedSurface` concept. This concept was however replaced by the `Appearance` module in CityGML 1.0 and therefore is marked deprecated. Although the CityGML specification disadvises the use of the `TexturedSurface` concept, it is still allowed even in CityGML 2.0 datasets. The Importer/Exporter can parse and interpret `TexturedSurface` information but will automatically convert this information losslessly into the `Appearance` module. Since `TextureSurface` information is not organized into themes but a theme is mandatory in the context of the `Appearance` module, the user has to define a *theme* that shall be used in the conversion process [2]. The default value is *rgbTexture*.

5.6.1.6 Geometry

Before importing the city objects into the 3D City Database, the Importer/Exporter can apply an affine coordinate transformation to all geometry objects. Per default, this option is disabled though.

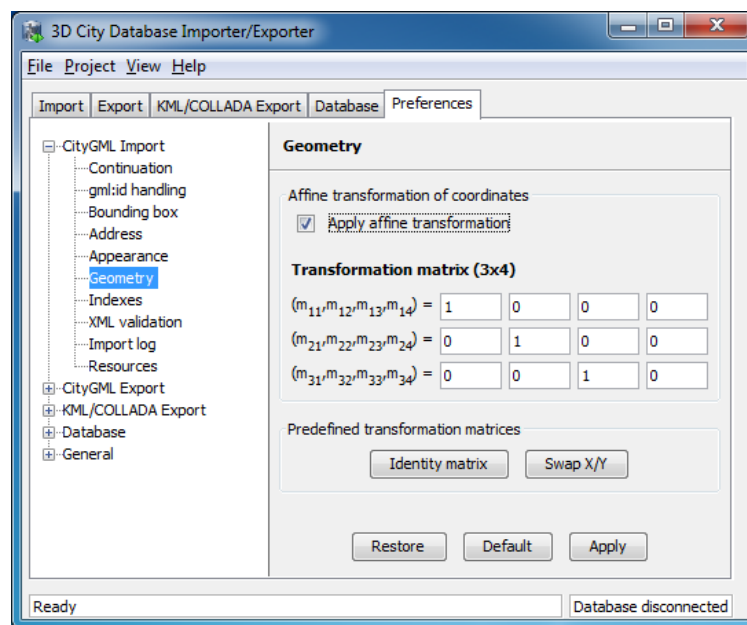


Figure 82: CityGML import preferences – Geometry.

An affine transformation is any transformation that preserves collinearity (i.e., points initially lying on a line still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). It will move lines into lines, polylines into polylines and polygons into polygons while preserving all their intersection properties. Geometric contraction, expansion, dilation, reflection, rotation, skewing, similarity transformations, spiral similarities, and translation are all affine transformations, as are their combinations.

The affine transformation is defined as the result of the multiplication of the original coordinate vectors by a matrix plus the addition of a translation vector.

$$\vec{p}' = A \cdot \vec{p} + \vec{b}$$

In matrix form using homogenous coordinates:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The coefficients of this matrix and translation vector can be entered in this preferences dialog (cf. Figure 82). The first three columns define any linear transformation; the fourth column contains the translation vector. The affine transformation does neither affect the dimensionality nor the associated reference system of the geometry object, but only changes its coordinate values. It is applied the same to all coordinates in all objects in the original CityGML file. This also includes all matrixes in CityGML like the 2x2 matrixes of `GeoreferencedTextures`, the 3x4 transformation matrixes of `TexCoordGen` elements used for texture mapping and the 4x4 transformation matrixes for `ImplicitGeometries`.

Note: An affine transformation cannot be undone or reversed after the import using the Importer/Exporter.

Two elementary affine transformations are predefined: 1) *Identity matrix* (leave all geometry coordinates unchanged), which serves as an explanatory example of how values in the matrix should be set, and 2) *Swap X/Y*, which exchanges the values of *x* and *y* coordinates in all geometries (and thus performs a 90 degree rotation around the *z* axis). The latter is very helpful in correcting CityGML datasets that have northing and easting values in wrong order.

Example: For an ordinary translation of all city objects by 100 meters along the *x*-axis and 50 meters along the *y*-axis (assuming all coordinate units are given in meters), the *identity matrix* must be applied together with the translation values set as coefficients in the translation vector:

$$\vec{p}' = \begin{bmatrix} 1 & 0 & 0 & 100 \\ 0 & 1 & 0 & 50 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \vec{p}$$

5.6.1.7 Indexes

In addition to the `Database` tab on the operations window, which lets you enable and disable spatial and normal indexes in the 3D City Database manually (cf. chapter 5.2.2), with this preference settings a default index strategy for database imports can be determined.

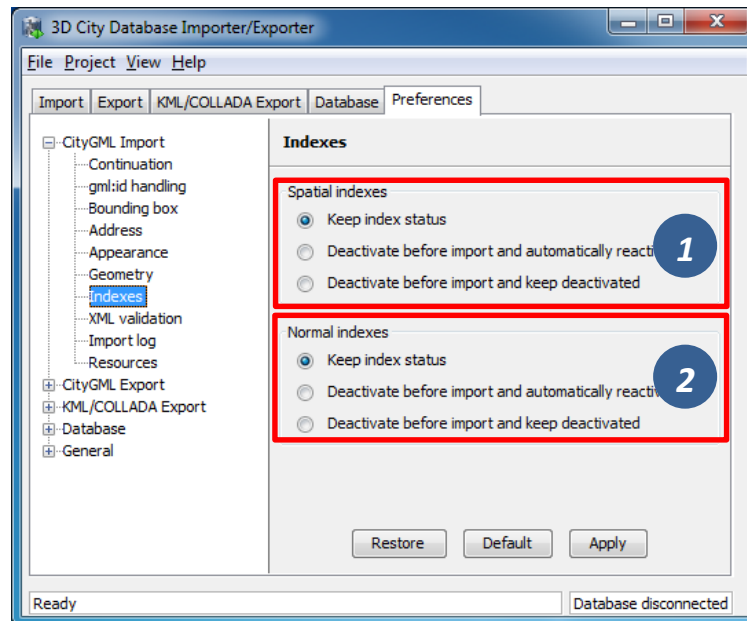


Figure 83: CityGML import preferences – Indexes.

The dialog differentiates between settings for *spatial indexes* [1] and *normal indexes* [2], but offers the same options for each index type.

The default setting is to not change the current status (i.e., either enabled or disabled) of the indexes. This default behavior can be changed so that indexes are always disabled before starting and import process. The user can choose whether the indexes shall be automatically reactivated after the import has been finished.

Note: All indexes are *enabled* after setting up a new instance of 3D City Database.

Note: It is *strongly recommended* to *deactivate the spatial indexes before running a CityGML import on a big amount of data* and to reactivate the spatial indexes afterwards. This way the import will typically be a lot faster than with spatial indexes enabled. The situation may be different if only a small dataset is to be imported. Deactivating normal indexes should however never be required.

Note: Activating and deactivating indexes can take a long time, especially if the database fill level is high. Note that the operation **cannot be aborted** by the user since this could result in an inconsistent database state.

5.6.1.8 XML validation

On the **Import** tab of the operations window, the CityGML input files to be imported into the database can be manually validated against the official CityGML XML Schemas. This preference dialog lets a user choose to perform XML validation automatically with every database import.

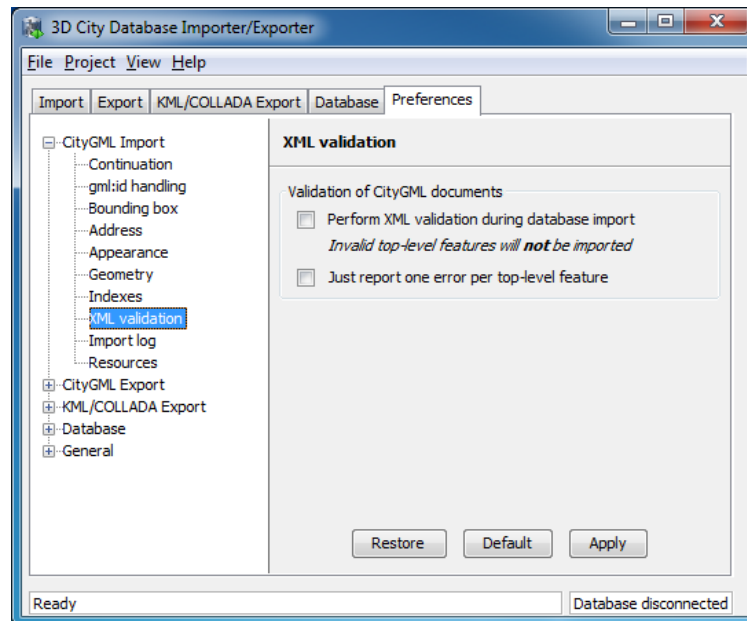


Figure 84: CityGML import preferences – XML validation.

In general, it is **strongly recommended** to ensure (either manually or automatically) that the input files are valid with respect to the CityGML XML schemas. Invalid files might cause the import procedure to behave unexpectedly or even to abort abnormally.

If XML validation is chosen to be performed automatically during imports, then every invalid top-level feature will be discarded from the import. Nevertheless, the import procedure will continue to work on the remaining features in the input file(s).

Validation errors are printed to the console window. Often, error messages quickly become lengthy and confusing. In order to keep the console output low, the user can choose to only report the first validation error per top-level feature and to suppress all subsequent error messages.

Note: The XML validation in general does not require internet access since the CityGML XML schemas are packaged with the Importer / Exporter. These internal copies of the official XML schemas will be used to check CityGML XML content in input files. The user cannot change this behavior. External XML schemas will only be considered in case of unknown XML content, which might require internet access. Precisely, the following rules apply:

- If an XML element's namespace is part of the official CityGML 2.0 or 1.0 standard, it will be validated against the internal copies of the official CityGML 2.0 or 1.0 schemas (no internet access needed).
- If the element's namespace is unknown, the element will be validated against the schema pointed to by the *xsi:schemaLocation* value on the root element or the element itself. This is necessary when, for instance, the input document contains XML content from a CityGML Application Domain Extension (ADE). Note that loading the schema might require internet access.

- If the element's namespace is unknown and the *xsi:schemaLocation* value (provided either on the root element or the element itself) is empty, validation will fail with a hint to the element and the missing schema document.

5.6.1.9 Import log

A CityGML import process not necessarily works on all CityGML features within the provided input file(s). An obvious reason for this are spatial or thematic filters that naturally narrow down the set of imported features. Also in case the import procedure aborts early (either requested by the user or caused by severe import errors), not all input features might have been processed. In order to understand which top-level were actually loaded into the database during an import session, the user can choose to let the Importer/Exporter create an *import log*.

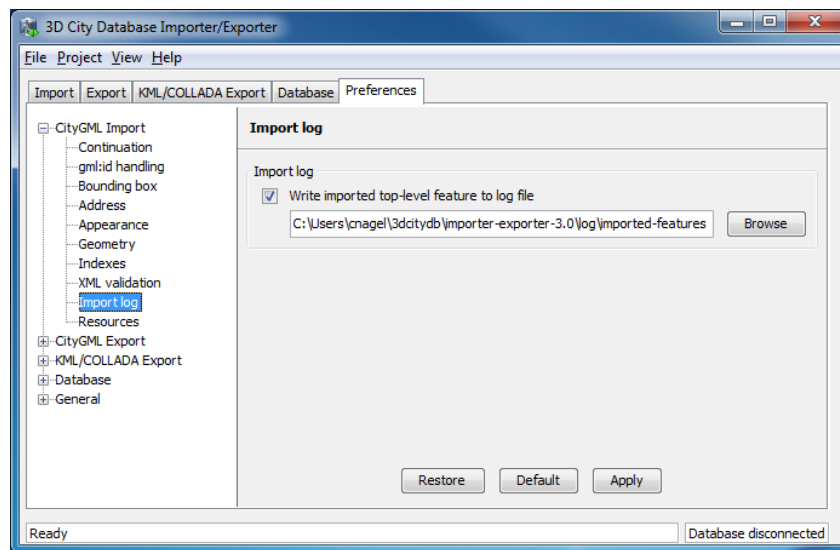


Figure 85: CityGML import preferences – Import log.

Simply enable the checkbox on this settings dialog to activate import logs (disabled per default). You additionally have to provide a folder where the import log files will be created in. Either type the folder name manually or use the *Browse* button to open a file selection dialog. The application proposes to use a folder within your user's home directory but this proposal can be overridden.

In order to easily relate import logs to different 3D City Database instances managed on the *Database* tab, the Importer/Exporter creates one subfolder for each connection entry below the folder provided in the settings dialog. The *description text* of the connection entry (cf. chapter 5.2.1) is used as folder name. Within that subfolder, a separate log file is created for every input file during an import to that 3D City Database connection. The filename includes the date and time of the import session according to following pattern:

imported-features-yyyy_MM_dd-HH_mm_ss_SSS.log

The import log is a simple CSV file with one record (line) per imported top-level feature. The following figure shows an example.


```

#3D City Database Importer/Exporter, version "3.0-b145"
#Imported top-level features from file: C:\test.gml
#Database connection string: citydb@localhost:5432/test
#Timestamp: 2014/10/21 23:18:59.414
FEATURE_TYPE,CITYOBJECT_ID,GML_ID_IN_FILE
BUILDING,46759,GEB_TH_Default_GEB_2034
BUILDING,46760,GEB_TH_Default_GEB_124
BUILDING,46763,GEB_TH_Default_GEB_1519
BUILDING,46768,GEB_TH_Default_GEB_1137
BUILDING,46772,GEB_TH_Default_GEB_1153
BUILDING,46776,GEB_TH_Default_GEB_1229
BUILDING,46779,GEB_TH_Default_GEB_1755
BUILDING,46783,GEB_TH_Default_GEB_1261
BUILDING,46791,GEB_TH_Default_GEB_1017
BUILDING,46799,GEB_TH_Default_GEB_1291
BUILDING,46804,GEB_TH_Default_GEB_1145
BUILDING,46808,GEB_TH_Default_GEB_1479
BUILDING,46815,GEB_TH_Default_GEB_1319
BUILDING,46821,GEB_TH_Default_GEB_1471
BUILDING,46825,GEB_TH_Default_GEB_1041
BUILDING,46828,GEB_TH_Default_GEB_1117
BUILDING,46831,GEB_TH_Default_GEB_1551
#Import successfully finished.

```

Figure 86: Example import log.

The first four lines of the import log contain metadata about the *version of the Import/Exporter* that was used for the import, the *absolute path to the CityGML input file*, the *database connection string*, and the *timestamp of the import*. Each line starts with # character in order to mark its content as metadata.

The first line below the metadata block provides a header for the fields of each record. The field names are `FEATURE_TYPE`, `CITYOBJECT_ID`, and `GML_ID_IN_FILE`. A single comma separates the fields. The records follow the header line. The meaning of the fields is as follows:

- `FEATURE_TYPE` An uppercase string representing the type of the imported CityGML feature.
- `CITYOBJECT_ID` The value of the ID column (primary key) of the CITYOBJECT table where the feature was inserted.
- `GML_ID_IN_FILE` The original `gml:id` value of the feature in the input file (might differ in database due to import settings).

The last line of each import log is a footer that contains metadata about whether the import was *successfully finished* or *aborted*.

5.6.1.10 Resources

Multithreading settings. The software architecture of the Importer/Exporter is based on multithreading. Put simply, the different tasks of an import process are carried out by separate threads. The decoupling of compute bound from I/O bound tasks and their parallel non-blocking processing usually leads to an increase of the overall application performance. For example, threads waiting for database response do not block threads parsing the input document or processing the CityGML input features. In a multi-core environment, threads can even be executed simultaneously on multiple CPUs or cores.

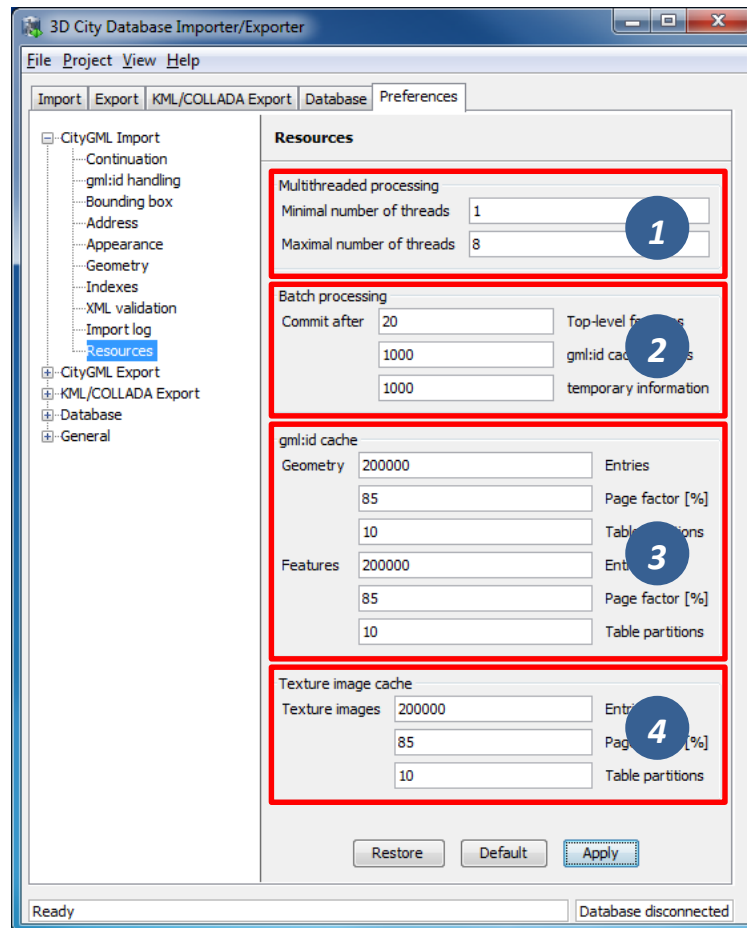


Figure 87: CityGML import preferences – Resources.

The `Resource` settings allow for controlling the minimum and maximum number of concurrent threads during import [1]. Make sure to enter reasonable values depending on your hardware configuration. By default, the maximum number is set to the number of available CPUs/cores times two. Before starting the import process, the minimum number of threads is created. Further threads up to the specified maximum number are only created if necessary.

Note: A higher number of threads *does not necessarily result in a better performance*. In contrast, a too high number of active threads faces disadvantages such as thread life-cycle overhead and resource thrashing. Also, note that each thread requires its *own physical connection to the database*. Therefore, your database must be ready to handle a sufficient number of parallel physical connections. Ask you database administrator for assistance.

Cache settings. The Importer/Exporter employs strategies for parsing CityGML datasets of arbitrary file size and for resolving XLink references. A naive approach for XLink resolving would read the entire CityGML dataset into main memory. However, CityGML datasets quickly become too big to fit into main memory. For this reason, the import process follows a two-phase strategy: In a first run, features are written to the database neglecting references to remote objects. If a feature contains an XLink though, any context information about the XLink is written to temporary database tables. This information comprises, for instance, the table name

and primary key of the referencing feature/geometry instance as well as the `gml:id` of the target object.

In addition, while parsing the document, the import process keeps track of every encountered `gml:id` as well as the table name and primary key of the corresponding object in database. It is important to record this information because a priori it cannot be predicted whether or not a `gml:id` is referenced by an XLink from somewhere else in the document. In order to ensure fast access, the information is cached in memory. If the maximum cache size is reached, the cache is paged to temporary database tables to prevent memory overflows. In a second run, the temporary tables containing the context information about XLinks are revisited and queried. Since the entire CityGML document has been processed at this point in time, valid references can be resolved and processed accordingly. With the help of the `gml:id` cache, the referenced objects can be quickly identified within the database.

The caching and paging behaviour for `gml:id` values can be influenced via the `Resource` preferences [3]. The dialog lets a user enter the maximum number of `gml:id` values to be held in main memory (default: 200,000 *entries*), the percentage of entries that will be written to the database if the cache limit is reached (*page factor*, default: 85%), as well as the number of parallel temporary tables used for paging (*table partitions*, default: 10). The Importer/Exporter employs different caches for `gml:id` values of geometries and features [3]. Moreover, a third cache is used for handling texture atlases and offers similar settings [4].

Batch settings. In order to optimize database response times, multiple database statements are submitted to the database in a single request (*batch processing*). This allows for an efficient data processing on the database side. The user can influence the number of SQL statements in one batch through the settings dialog [2]. The dialog differentiates between batch sizes for CityGML features (default: 20) and `gml:id` caches respectively temporary XLink information (default: 1000 each).

Note: All database operations within one batch are buffered in main memory before being submitted to the database. Thus, the Importer/Exporter might run out of memory if the batch size is too high. After a batch is submitted, the transaction is committed.

5.6.2 CityGML export preferences

5.6.2.1 CityGML version

The `CityGML version` preference settings let you choose the target CityGML version when exporting 3D city model content from the database to a CityGML dataset.

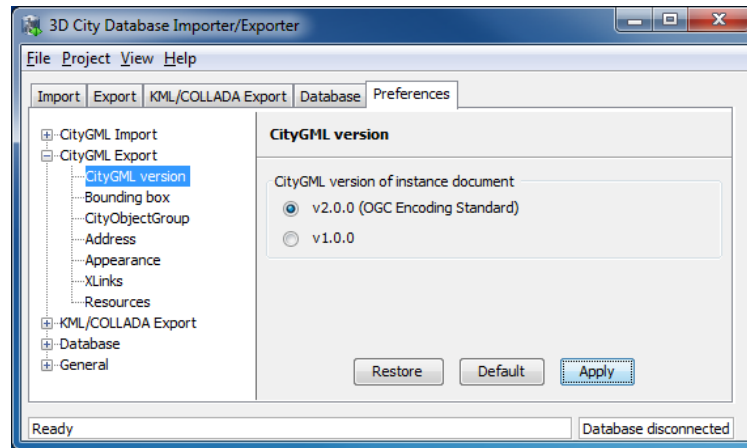


Figure 88: CityGML export preferences – CityGML version.

The default value is *CityGML version 2.0.0*, which is the current version of the OGC CityGML Encoding Standard. In addition, also the preceding *version 1.0.0* is still supported.

Note: CityGML 2.0.0 introduces new feature types such as bridges and tunnels that are not available in CityGML 1.0.0. If the 3D City Database instance contains features of these types, they *will be neglected* in an export to CityGML version 1.0.0 simply because they cannot be encoded in this version.

5.6.2.2 Bounding box

When exporting the entire 3D city model content stored in the 3D City Database to a single CityGML instance document, the resulting file easily becomes very large. Although the Importer/Exporter supports writing files of arbitrary size (only limited by the file system of the operating system), such files may be too big to be processed by other applications. Furthermore, if the 3D city model is fully textured, then the number of texture files exported into the same subfolder may become very high. This, in turn, may adversely affect the file access time.

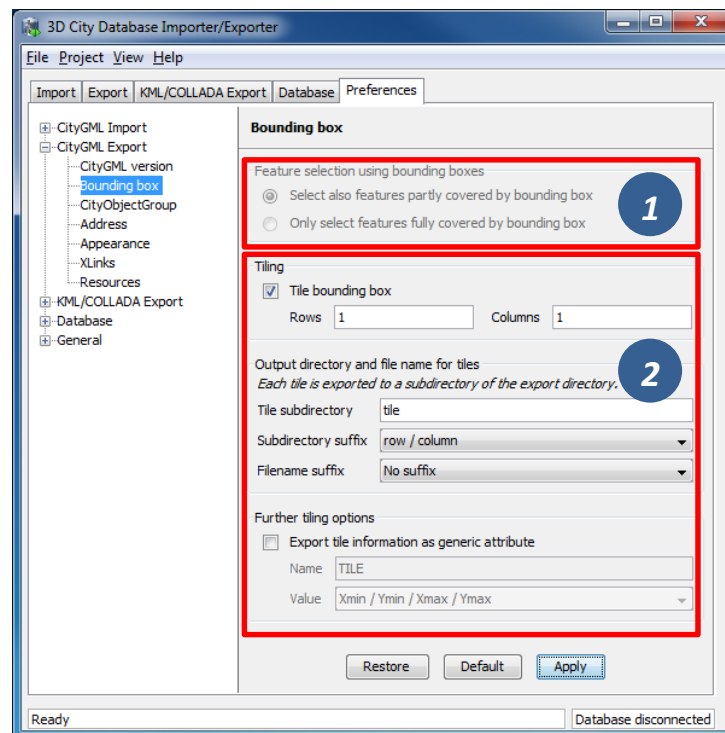


Figure 89: CityGML export preferences – Bounding box.

The Importer/Exporter allows for applying a spatial *bounding box filter* to CityGML exports which helps in tailoring the export to a geographic area of interest or in reducing the number of exported features and thus of the resulting file size [1]. The bounding box is provided on the `Export` tab of the operations window (cf. chapter 5.4) and is evaluated against the `ENVELOPE` column of the `CITYOBJECT` table. Per default, top-level features whose bounding box is inside or overlaps with the filter geometry are exported [1]. Alternatively, the user can choose a more restrictive filter setting to only consider those features whose bounding box is completely inside the filter geometry.

If, however, the area of interest is still comparatively large, the Importer/Exporter can automatically tile the area specified by the bounding box filter [2]. Tiled exports are only available with the bounding box filter being enabled on the `Export` tab (further filters can of course additionally be used). To make use of the tiling feature, please check the *Tile bounding box* option. Checking this option automatically disables the user dialog in [1].

When tiling is selected, the number of rows and columns of the (resulting gridded) export must be specified. Space is distributed evenly among them so that all rows have the same height and all columns the same width. To decide whether a feature has to be exported within a specific tile, the center of its envelope has to be either inside or on the left or top border of the tile.

During export, a subfolder below the main export folder specified on the `Export` tab (cf. chapter 5.4) will be created for each tile. Each folder name consists of a prefix and a tile-specific suffix [2]. The suffix may contain the row and column number of the tile exported or a combination of the tile's minimum / maximum coordinates. If a coordinate suffix is chosen, the coordinates will be given in the reference system specified for the CityGML export (cf. chapter 5.4; default value is the internal SRS of the 3D City Database instance), even if the coordinates

of the bounding box filter are given in another user-defined SRS. This makes it easy to relate objects to tiles since the coordinates of the objects contained in the tile are exported in the same reference system. The filename of the CityGML instance document created in each subfolder corresponds to the one defined on the `Export` tab. However, a tile-specific suffix may be appended [2].

For further traceability, it is possible to attach a generic String attribute called *TILE* to each exported CityGML feature, indicating which tile it belongs to. The options for identifying the tile (the value of this generic attribute) are the same as for the tile directory suffix.

Note: If the entire 3D city model stored in the 3D City Database instance shall be exported with the tiling feature enabled, a bounding box spanning the overall area of the model has to be provided. This bounding box can be easily calculated on the `Database` tab (cf. chapter 5.2.2).

Note: Using the center of an object's envelope as criterion for a tiled export has a side-effect when tiling is combined with the *counter filter* on the `Export` tab: the number of objects being exported can no longer be exactly determined since the calculation of the object's envelope center has to happen at a later point in time. Therefore, the *counter filter* only sets a possible maximum number in this filter combination. For instance, when setting *from #1 to #500* on the *counter filter*, only 493 features may be exported in case the missing seven objects are discarded after being queried due to their envelope center not being within the tile. This is a correct behavior, so the Importer/Exporter will not report any errors.

5.6.2.3 CityObjectGroup

When exporting city object groups, also group members are written to the target CityGML dataset (cf. chapter 5.4). Per default, group members are given *by value* (i.e., inline the group feature) and only group members satisfying the export filter settings are considered.

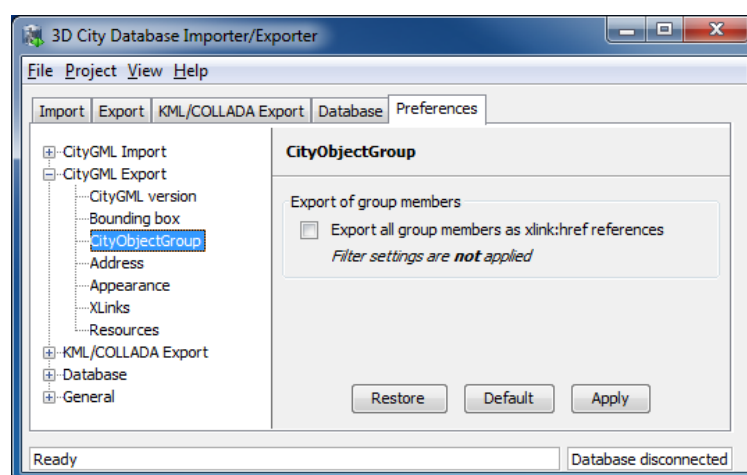


Figure 90: CityGML export preferences – CityObjectGroup.

The default behavior can be changed using this preference dialog. When checking the option *Export all group members as xlink:href references*, group members are instead given by

reference using GML's XLink mechanism. This may increase the performance of the CityGML export.

Note: When exporting group members by reference, then export filter settings *are not applied*. Instead, an *xlink:href* reference is created *for each and every group member* even if it is not contained in the resulting CityGML dataset.

5.6.2.4 Address

Similar to the import of xAL address information (see chapter 5.6.1.4), the user can choose how address information should be exported to a target CityGML dataset. The available options of the Address export preferences are shown in the figure below.

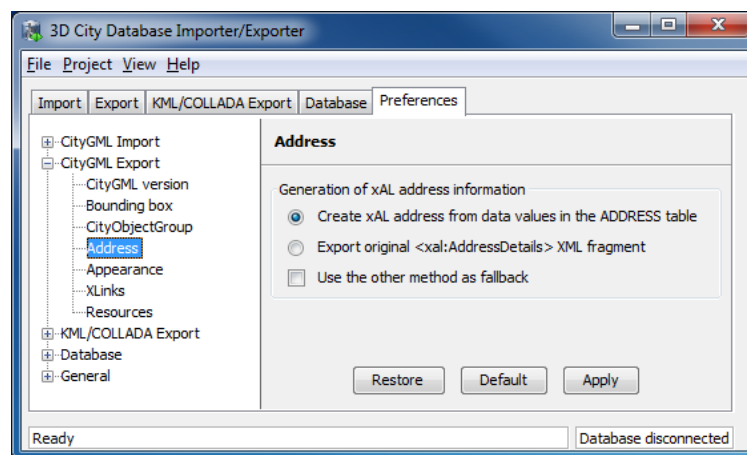


Figure 91: CityGML export preferences – Address.

Address information is exported from the data values in the ADDRESS table of the 3D City Database instance. As discussed in chapter 5.6.1.4, these values may however lack data present in the original xAL fragment or they may even contain no data at all when the address information differs too much from the supported xAL templates (cf. Figure 79). In such cases, using the original `<xal:AddressDetail>` element stored in the XAL_SOURCE column is the only means to achieve a lossless reconstruction of the initial address data.

Since importing the original `<xal:AddressDetail>` fragment into XAL_SOURCE does not hinder the population of the remaining columns of the ADDRESS table (STREET, HOUSE_NUMBER, etc.), there are two possible ways to reconstruct the address contents when exporting from the 3D City Database.

- 1) The default option is to build the xAL address from the columns of the ADDRESS table *without considering* the XAL_SOURCE column. In this case, the XML encoding of the xAL address follows the first template as shown Figure 79.
- 2) Optionally, the xAL fragment is taken “as is” from the XAL_SOURCE column and inserted literally into the target CityGML document. This way there will be no loss of information and the address encoding will be identical to the original source datasets. Obviously, this option requires that the XAL_SOURCE column has been populated during import (chapter 5.6.1.4).

Both options are mutually exclusive but one can be used as a fallback alternative to the other if the first chosen renders no results.

5.6.2.5 Appearance

The Appearance export preferences are similar to the settings available for importing CityGML (cf. chapter 5.6.1.5).

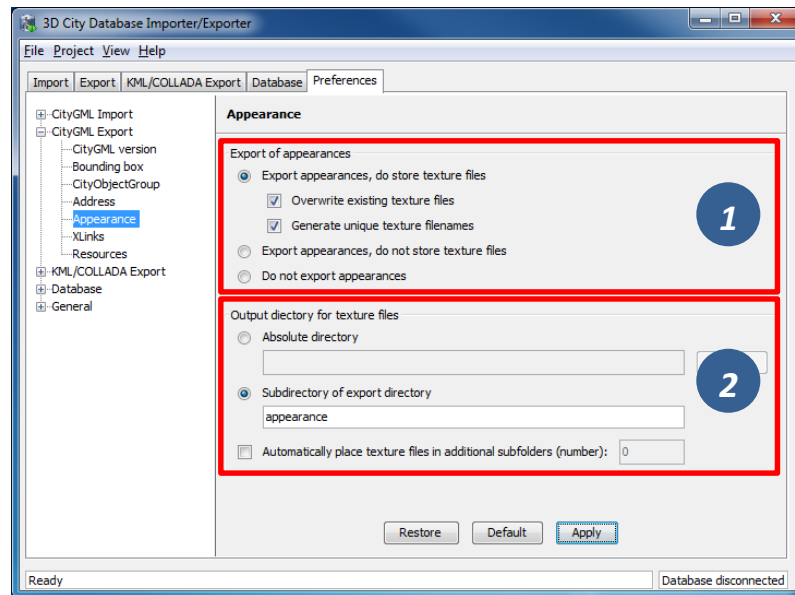


Figure 92: CityGML export preferences – Appearance.

Per default, both appearance information and texture image files associated with the city objects in the 3D City Database are exported to the target CityGML dataset [1]. Alternatively, the user can choose to only export appearance information without writing texture images or to even suppress any appearance information.

When exporting texture files, the two additional options *Overwrite existing texture files* and *Generate unique texture filenames* influence the way in which texture files are written to the file system [1].

1) *Overwrite existing texture files*

Texture files are stored in a separate folder of the file system. Before exporting a texture image file into this folder, the Importer/Exporter can check whether a file of the same filename already exists in this folder. In this case, the existing file will be kept if this option is *not enabled*. Otherwise, and per default, there is no check and a texture file of the same name will be overwritten (if it exists).

2) *Generate unique texture filenames*

Often filenames for texture images are automatically created from a naming scheme involving some counter (e.g., a prefix “*tex*” followed by a number incremented by 1 for each new image). It thus can happen that two city objects within the same or different instance documents are assigned a texture image file of the same name but with different content (e.g., if the texture files are distributed over several folders). In the 3D City Database, texture images are stored in separate records and thus duplicate filenames are

not an issue. When exporting to CityGML however, two texture files of the same name might be written to the same target folder, in which case one is replaced with the other. This will obviously lead to false visualizations and issues in workflows consuming the exported CityGML data. For this reason, checking this option (default) will force the export process to generate unique filenames for each texture file based on the primary key value of the `TEX_IMAGE` table. Therefore, the filename even keeps stable amongst several exports from the 3D City Database.

The location where to store the texture files can be defined by the user [2]. The default option is to pick a folder below the export directory and thus relative to the target CityGML file. The default folder name is “*appearance*”. Instead of a local path, also an absolute path can be provided. In this case, the same folder will be used in subsequent exports from the 3D City Database.

Especially when using Windows, placing a large number of files into the same folder might lead to severe time lags when trying to access files in this folder or to write new files to this folder. This might negatively affect the performance for large exports. For this reason, the Importer/Exporter can automatically distribute the texture files in additional subfolders that are automatically created. Simply check the option *Automatically place texture files in additional subfolders* and provide the number of subfolder to be used.

5.6.2.6 XLinks

Both the 3D City Database and the Importer/Exporter are capable of handling XLinks. If the CityGML input document that is imported into the 3D City Database contains XLink references to features and/or geometries, then this information is kept in the database in order to be able to reconstruct the XLinks upon database export. This is also the default behavior.

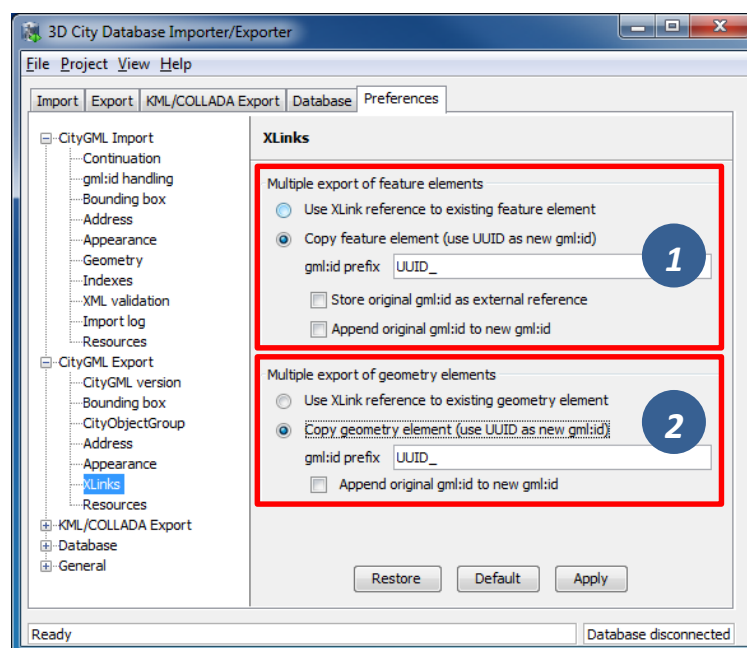


Figure 93: CityGML export preferences – XLinks.

Depending on the target application that consumes the exported CityGML dataset, this default behavior may be disadvantageous, especially if the target application cannot follow and resolve

XLink references. In such cases, the XLinks preference settings let a user change the default behavior so that the referenced objects are exported *by value* rather than *by reference*. Put differently, instead of an XLink reference, a copy of the original feature or geometry is placed into the CityGML dataset. This necessarily requires that the `gml:id` of the copy is different from the `gml:id` of the original object because identical `gml:id` values are not allowed in the same dataset. The Importer/Exporter takes care of this issue and creates new `gml:id` values for the copies based on UUID values.

The user can define the behavior for exporting XLinks differently for features [1] and geometries [2]. The settings allow to provide a *prefix* string that will be used when creating new `gml:id` values (default: “*UUID_*”). In addition, the original `gml:id` may be appended to the newly created one. Whereas these settings are available for both features and geometries, the user can additionally choose to create a CityGML `<ExternalReference>` element for features that carries the original `gml:id` value and to attach this external reference as attribute to the copied feature.

5.6.2.7 Resources

Just like with CityGML imports, the export process is implemented based on multithreaded data processing in order to increase the overall application performance. Likewise, in order to reconstruct XLinks during exports (cf. chapter 5.6.2.6), the export process also needs to keep track of each and every `gml:id` of exported features and geometry objects. For fast access, the `gml:id` values are kept in main memory and are only paged to temporary database tables in case the predefined cache size limit is reached.

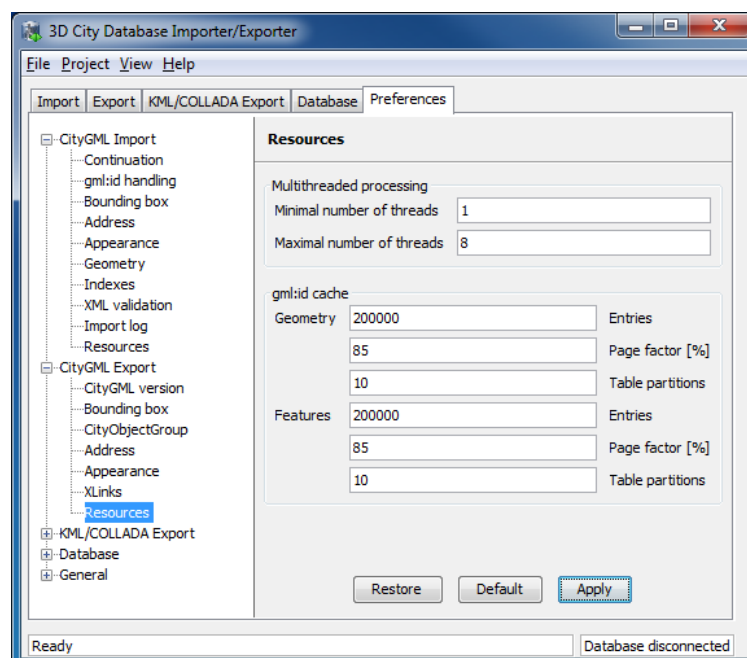


Figure 94: CityGML export preferences – Resources.

The Resource preferences allow for setting the number of *concurrent threads* to be used in the export process and for defining the *sizes* and *page factors* of the `gml:id` caches for features

and geometries. The meaning of the values is identical to the `Resource` preferences for CityGML imports. So please refer to chapter 5.6.1.10 for more details.

5.6.3 KML/COLLADA export preferences

The preferences tab contains four subnodes – *General*, *Rendering*, *Balloon*, and *Altitude/Terrain* – make customization of these exports possible. These settings will be explained in the following sections in details.

5.6.3.1 General Preferences

Some common features of the exported files, especially those related to tiling options, can be set under the preferences tab, node *KML/COLLADA Export*, subnode *General*.

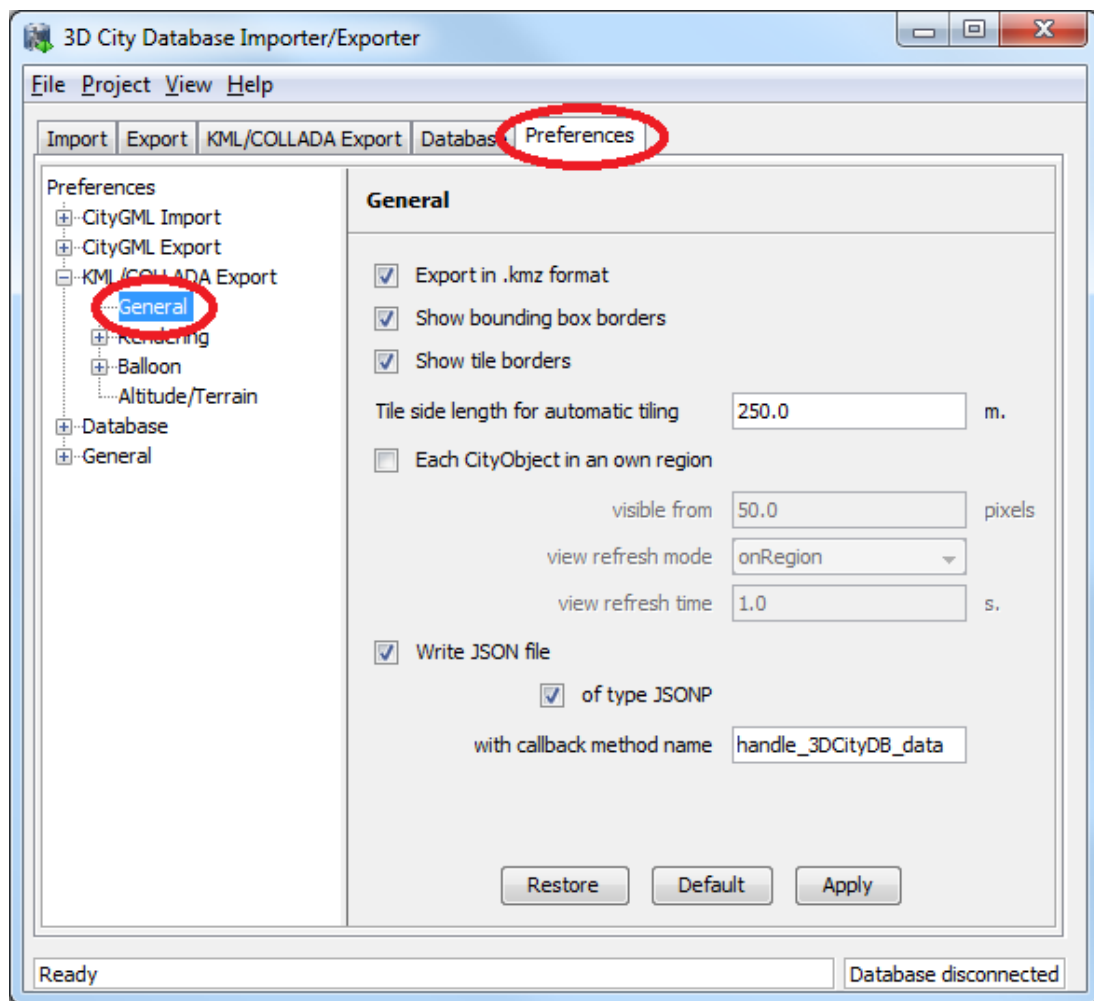


Figure 95: General settings for the KML/COLLADA export.

Export in kmz format

Determines in which format single files and tiled exports should be written: kmz when selected, kml when not. Whatever format is chosen, the main file (so called master file, pointing to all others) will always be a kml file, all other files will comply with this setting.

Tests have shown shorter loading times (in Google Earth) for the kml format (as opposed to kmz) when loading from the local hard disk. The Earth Browser's stability also seems to improve when using the uncompressed format. On the other hand, when loading files from a

server kmz reduces the amount of requests considerably, thus increasing performance. Kmz is also recommended for a better overview since kml exports may lead to a large number of directories and files.

Show bounding box borders

When exporting a region of interest via the bounding box option in the *KML/COLLADA Export* tab, this checkbox specifies whether the borders of the whole bounding box will be shown or not. The frame of the bounding box is four times thicker than the borders of any single tile in a tiled export.

Show tile borders

Specifies whether the borders of the single tiles in a tiled export will be shown or not.

Tile side length for automatic tiling

Applies only to automatically tiled exports and sets the approximate square size of the tiles. Since the Bounding Box settings in the *KML/COLLADA Export* tab are the determining factor for the area to be exported and have priority over this setting, the resulting tiles may not be perfectly square or have exactly the side length fed into this field.

Each CityObject in an own region

The visibility of the objects exported can be further fine-tuned by this option. While the visibility settings on the main *KML/COLLADA Export* tab apply to the whole area (*no tiling*) or to each tile (*automatic, manual*) being exported, this checkbox allows to individually define a KML `<Region>` for every single city object. The limits of the object's region are those of the object's CityGML Envelope.

Following the KML Specification [Wilson 2008], each KML `<Region>` is defined inside a KML `<NetworkLink>` and has an associated KML `<Link>` pointing to a file. This implies when this option is chosen a subfolder is created for each object exported, identified by the object's gmlId. The object's subfolder will contain any KML/COLLADA files needed for the visualization of the object in the Earth browser. This folder structure (which can contain a large number of subfolders) is required for the KML `<Region>` visibility mechanism to work.

When active, the parameters affecting the visibility of the object's KML `<Region>` can be set through the following related fields.

The field *visible from* determines from which size on screen the object's KML `<Region>` becomes visible, regardless of the visibility value of the containing tile, if any. Since this value is the same for every single object and they have all different envelope sizes a good average value should be chosen.

The field *view refresh mode* specifies how the KML `<Link>` corresponding to the KML `<Region>` is refreshed when the geographic view changes. May be one of the following:

- **never** - ignore changes in the geographic view.
- **onRequest** - refresh the content of the KML `<Region>` only when the user explicitly requests it.

- **onStop** - refresh the content of the KML <Region> *n* seconds after movement stops, where *n* is specified in the field *view refresh time*.
- **onRegion** - refresh the content of the KML <Region> when it becomes active.

As stated above, the field *view refresh time* specifies how many seconds after movement stops the content of the KML <Region> must be refreshed. This field is only active and its value is only applied when *view refresh mode* is onStop.

Write JSON file

After exporting some cityobjects in KML/COLLADA you may need to include them into websites or somehow embed them into HTML. When working with tiled exports referring to a specific object inside the KML/COLLADA files can become a hard task if the contents are loaded dynamically into the page. It is impossible to tell beforehand which tile contains which object. This problem can be solved by using a JSON file that is automatically generated when this checkbox is selected.

In the resulting JSON file each exported object is listed, identified by its gmlId acting as a key and some additional information is provided: the envelope coordinates in CRS WGS84 and the tile, identified by row and column, the object belongs to. For untiled exports the tile's row and column values are constantly 0.

This JSON file has the same name as the so-called master file and is located in the same folder. Its contents can be used for indexed search of any object in the whole KML/COLLADA export.

JSON file example:

```
{
  "BLDG_0003000b0013fe1f": {
    "envelope": [13.411962, 52.51966, 13.41277, 52.520091],
    "tile": [1, 1]},
  [...]
  "BLDG_00030009007f8007": {
    "envelope": [13.406815, 52.51559, 13.40714, 52.51578],
    "tile": [0, 0]}
}
```

The JSON file can automatically be turned into JSONP (JSON with padding) by means of adding a function call around the JSON contents. JSONP provides a method to request data from a server in a different domain, something typically forbidden by web browsers since it is considered a cross-site-scripting attack (XSS). Thanks to this minimal addition the JSON file contents can be more easily embedded into webpages or interpreted by web kits without breaking any rules. The function call name to be added to the original JSON contents is arbitrary and must only be entered in the callback method name field.

5.6.3.2 Rendering Preferences

Most aspects regarding the look of the KML/COLLADA exports when visualized in virtual globe like Google Earth can be customized under the preferences tab, node *KML/COLLADA Export*, subnode *Rendering*. Each of the top-level feature categories has its own *Rendering* settings. For the sake of clarity the most complex *Rendering* settings for *Buildings* will be here explained as an example. Settings for all other top-level features are either identical or simpler. An exceptional case is *GenericCityObject* which can be exported into point or line geometries, and the corresponding settings will be explained at the end of this section.

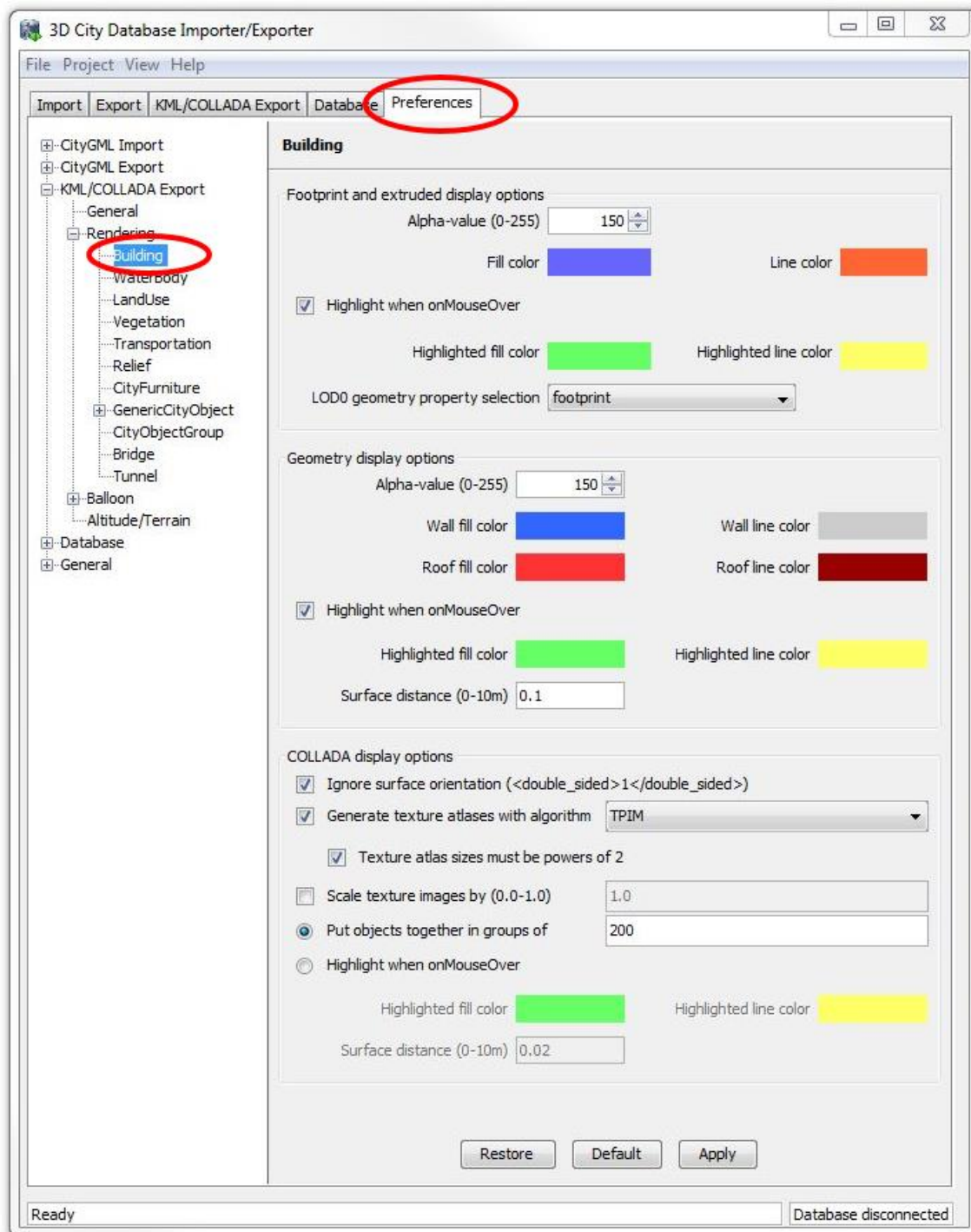


Figure 96: Rendering settings for the KML/COLLADA *Building* export.

All settings in this menu are grouped according to the display form they relate to.

Footprint and extruded display options

In this section the fill and line colors can be selected. Additionally, it can be chosen whether the displayed objects should be highlighted when being run over with the mouse or not. Highlighting colors can only be set when the highlighting option is enabled. The alpha value affects the transparency of all colors equally: 0 results in transparent (invisible) colors, 255 in completely opaque ones. A click on any color box opens a color choice dialog.

As defined in the CityGML specification [Gröger et al. 2012] CityGML version 2.0.0 allows LoD0 representation (footprint and roofprint representations) for buildings and building parts. If LoD0 in the Level of Export setting on the main *KML/COLLADA Export* tab is selected, there are three options available for LoD0 geometry export:

- *footprint*: the footprint geometries of the buildings or building parts will be exported
- *roofprint*: the roofprint geometries of the buildings or building parts will be exported
- *roofprint, if none then footprint*: footprint geometries will be exported if none of the roofprint geometries are found.

Geometry display options

This parameter section distinguishes between roof and wall surfaces and allows the user to color them independently. The alpha value affects the transparency of all roof and wall surface colors in the same manner as in the footprint and extruded cases: 0 results in transparent (invisible) colors, 255 in completely opaque ones. A click on any color box opens a color choice dialog.

As previously stated: when not explicitly modeled, thematic surfaces will be inferred for LoD1 or LoD2 based exports following a trivial logic (surfaces touching the ground –that is, having a lowest z-coordinate- will be considered wall surfaces, all other will be considered roof surfaces), in LoD3 or LoD4 based exports surfaces not thematically modeled will be colored as wall surfaces.

The highlighting effect when running with the mouse over the exported objects can also be switched on and off. Since the highlighting mechanism relies internally on a switch of the alpha values on the highlighting surfaces, the alpha value set in this section does not apply to the highlighted style of geometry exports, only to their normal style. For a detailed explanation of the highlighting mechanism see the following section.

COLLADA display options

These parameters control the export of textured models. The first option addresses the fact that sometimes objects may contain wrongly oriented surfaces (points ordered clockwise instead of counter-clockwise) as a result of errors in some previous data gathering or conversion process. When rendered, wrongly oriented surfaces will only be textured on the inside and become transparent when viewed from the outside. Ignore surface orientation informs the viewer to disable back-face culling and render all polygons even if some are technically pointing away from the camera.

Note: This will result in lowered rendering performance. Correcting the surface orientation data is the recommended solution. This option only provides a quick fix for visualization purposes.

Surface textures can be stored in an image file each, or grouped into large canvases containing all images clustered together, so called texture atlases, that significantly increase loading speed. Grouping images in an atlas or not and the algorithm selected for the texture atlas construction (differing in generation speed and canvas efficiency) can be set here. Depending on the algorithm and size of the original textures an object can have one or more atlases, but atlases are not shared between separate objects.

The texture atlas algorithms address the problem of two dimensional image packing, also known as 'knapsack problem', in different ways (see [Coffman et al. 1980]):

- **BASIC:** recursively divides the texture atlas into empty and filled regions (see <http://www.blackpawn.com/texts/lightmaps/default.html>). The first item is placed in the top left corner. The remaining empty region is split into two rectangles along the sides of the item. The next item is inserted into one of the free rectangles and the remaining empty space is split again. Doing this in a recursive way builds a binary tree representing the texture atlas. When adding an item, there is no information of the sizes of the items that are going to be packed after this one. This keeps the algorithm simple and fast. The items may be rotated when being inserted into the texture atlas.
- **TPIM:** touching perimeter (see [Lodi et al. 1999] and [Lodi et al. 2002]). Sorts images according to non-increasing area and orients them horizontally. One item is packed at a time. The first item packed is always placed in the bottom-left corner. Each following item is packed with its lower edge touching either the bottom of the atlas or the top edge of another item, and with its left edge touching either the left edge of the atlas or the right edge of another item. The choice of the packing position is done by evaluating a score, defined as the percentage of the item perimeter which touches the atlas borders and other items already packed. For each new item, the score is evaluated twice, for the two item orientations, and the highest value is selected.
- **TPIM w/o image rotation:** touching perimeter without rotation. Same as TPIM, but not allowing for rotation of the original images when packing. Score is evaluated only once since only one orientation is possible.

From the algorithms, *BASIC* is the fastest (shortest generation time) and produces good results, whereas *TPIM* is the most efficient (highest used area/total atlas size ratio).

Scaling texture images is another means of reducing file size and increasing loading speed. A scale factor of 0.2 to 0.5 often still offers a fairly good image quality while it has a major positive effect on these both issues. Default value is 1.0 (no scaling). This setting is independent from the atlas setting and both can be combined together. It is possible to generate atlases and then scale them to a smaller size for yet shorter loading times in Google Earth.

Buildings can be put together in groups into a single model/placemark. This can also speed up loading, however it can lead to conflicts with the digital terrain model (DTM) of the Earth browser, since buildings grouped together have coordinates relative to the first building on the

group (taken as the origin), not to the Earth browser's DTM. Only the first building of the group is guaranteed to be correctly placed and grounded in the Earth browser. If the objects being grouped are too far apart this can result in buildings hovering over or sinking into the ground or cracks appearing between buildings that should go smoothly together.

Up to Google Earth 7, no highlighting of model placemarks loaded from a location other than Google Earth's own servers is supported natively (glowing blue on mouse over). Therefore a highlighting mechanism of its own was implemented in the KML/COLLADA exporter: highlighting is achieved by displaying a somewhat "exploded" version of the city object being highlighted around the original object itself. "Exploded" means all surfaces belonging to the object are moved outwards, displaced by a certain distance orthogonally to the original surface. This "exploded" highlighting surface is always present, but not always visible: when the mouse is not placed on any building (or rather, on the highlighting surface surrounding it closely) this "exploded" highlighting surface has a normal style with an alpha value of 1, invisible to the human eye. When the mouse is placed on it, the style changes to highlighted, with an alpha value of 140 (hard-coded), becoming instantly visible, creating this model placemark highlighted feel.

The displacement distance for the exploded highlighting surfaces can be set here. Default value is 0.75m.

This highlighting mechanism has an important side effect: the model's polygons will be loaded and displayed twice (once for the representation itself, once for the highlighting), having a negative impact in the viewing performance of the Earth browser. The more complex the models are, the higher the impact is. This becomes particularly noticeable for models exported from a LoD3 basis upwards. The highlighting and grouping options are mutually exclusive.



Figure 97: Object exported in the COLLADA display form being highlighted on mouseOver.

GenericCityObject

In contrast to other top-level feature class, not only surface-based but also point and line geometries of *GenericCityObject* can be exported. Its *rendering* settings contains further two subnodes – “*Surface and Solid*” and “*Point and Curve*” – makes customization of these exports possible. Settings for “*Surface and Solid*” are either identical or simpler compared to other top-level features. “*Point and Curve*” settings will be explained in the followings.

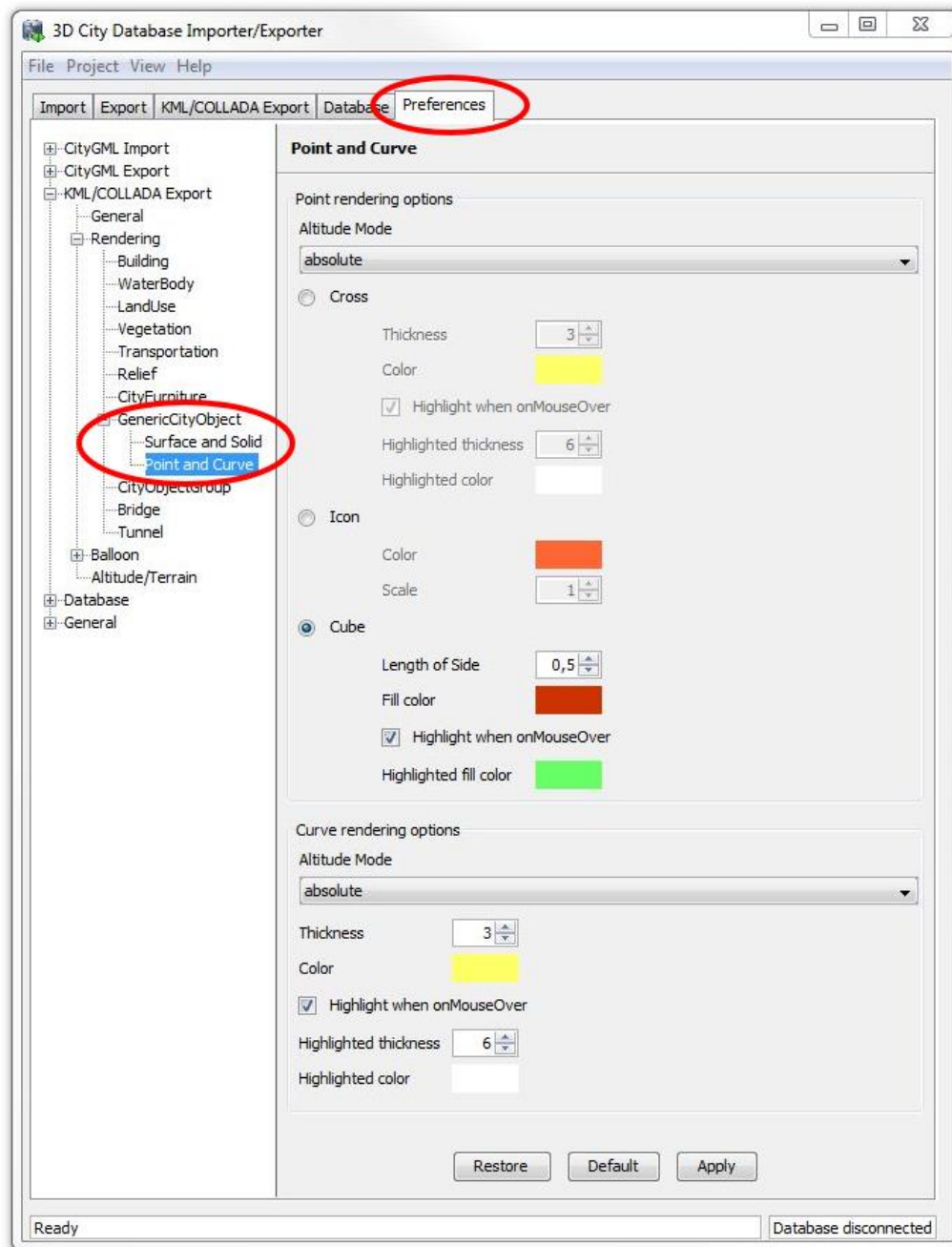


Figure 98: Rendering settings for the KML/COLLADA *GenericCityObject* export.

The field *Altitude mode* specifies how the coordinates of the point and curve geometries are interpreted. Possible value may be one of the following:

- **relative:** the altitude is interpreted as an absolute height value in meters according to the vertical reference system (EGM96 geoid in KML).

- **absolute**: the altitude is interpreted as a value in meters above the terrain.
- **clamp to ground**: the altitude is ignored and the point and curve geometry will be always clamp to the ground (= the relative altitude value equal to 0).

Three setting options are available which allow user to choose a more appropriate display form for point geometry on the 3D map:

- **Cross**: The point geometry can be spatially represented by a cross-line in the form like “X”. The user will change the thickness setting which affects the width of the cross-line geometry in pixels, and change the display color to be more visible. Highlighting option is also provided as well. The display settings for color and thickness of highlighting geometry are activated when the highlighting option is enabled.



Figure 99: point geometry displayed as cross-line.

- **Icon**: Another alternative way to display point geometry is using the KML-icon which is typically used to display point placemark for Google Earth. The Color of the icon can be determined by the user, and the scale option offers the possibility to specify the scaling factor of the icon for adjusting the display size of the icon. Default value is 1.0 (no scaling).



Figure 100: point geometry displayed as icon.

- **Cube:** It is also possible to represent the point geometry as a tiny solid particle whose central point is essentially identical to the represented point. Settings such as size, color, and highlighting are also provided to achieve the appropriate display effect.

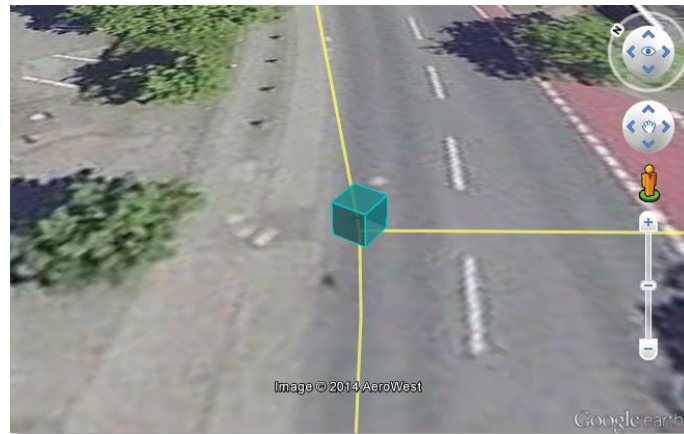


Figure 101: point geometry displayed as cube.

Settings for curve geometry are identical to those previously described in the part of “Cross” option for point geometry export.

5.6.3.3 Information Balloon Preferences

KML offers the possibility of enriching its placemark elements with information bubbles, so-called balloons, which pop up when the placemark is clicked on. This is supported by the Import/Export tool regardless of the display form the object is exported in.

Note: When exporting in the COLLADA display form it is recommended to enable the "highlighting on mouseOver" option, since model placemarks not coming from Google Earth servers are not directly clickable, but only through the sidebar. Highlighting geometries are, on the contrary, directly clickable wherever they are loaded from.

The contents of the balloon can be taken from a generic attribute called *Balloon_Content* associated individually to each city object in the 3DCityDB. They can also be uniform for all objects in an export by using an external HTML file as a template, or a combination of both: individually and uniformly set, the *Balloon_Content* attribute (individually) having priority over the external HTML template file (uniform). A few Balloon HTML template files can be found after software installation in the subfolder `templates/balloons` of the installation directory.

The balloons can be included in the doc.kml file generated at export, or they can be put into individual files (one for each object) written together into a "balloon" directory. This makes later adaption work easier if some post-processing (manual or not) is required. When balloon contents are put into a separate file for each exported object, access to local files and personal data must be granted in Google Earth (Tools → Options → General) for the balloons to show.

Balloon preferences can be set independently for each CityGML top-level feature type. That means every object can have its own individual template file (so that for instance, *WaterBody*

balloons display a different background image as *Vegetation* balloons), and it is perfectly possible to have information bubbles for some object types while some others have none. For *GenericCityObject*, the point and line geometry object can also has its own individual balloon setting. The following example is set around *Building* balloons but it applies exactly the same for all feature classes.

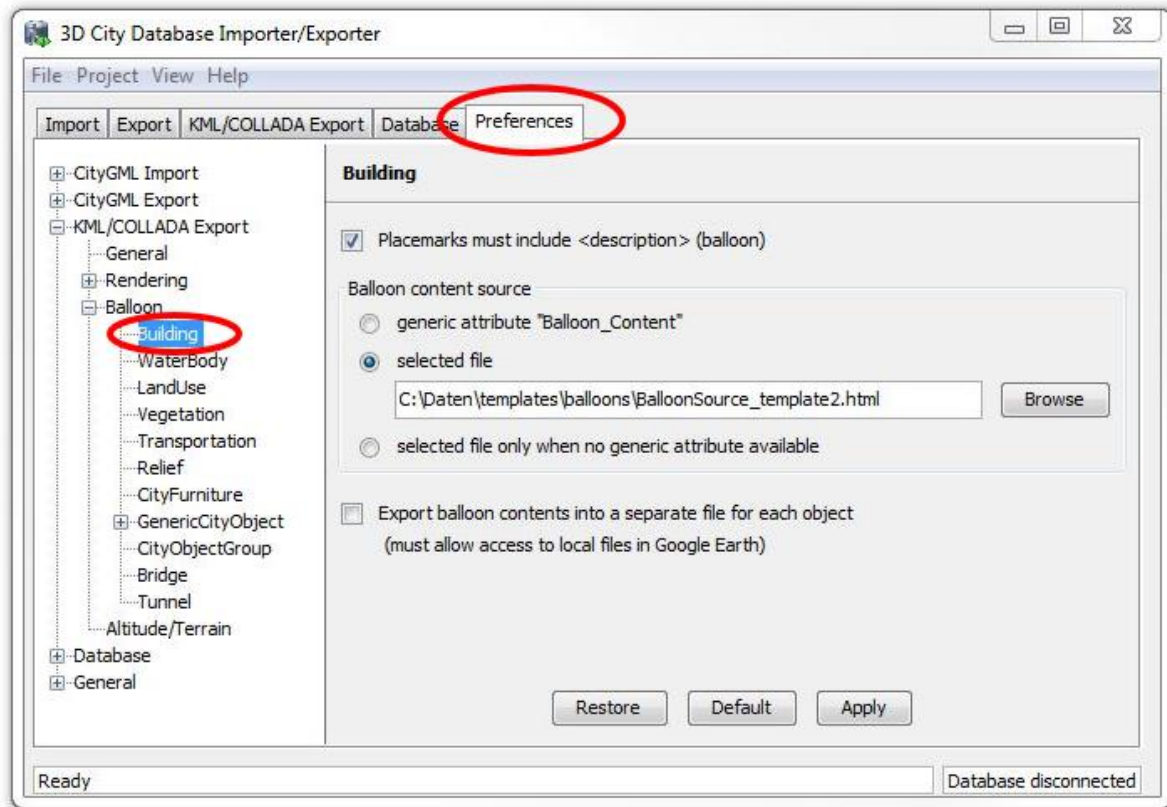


Figure 102: *Building* Balloon settings.

The balloon contents do not need to be static. They can contain references to the data belonging to the city object they relate to. These references will be dynamically resolved (i.e.: the actual value for the current object will be put in their place) at export time in a way similar to how Active Server Pages (ASP) [Microsoft, 2015] work. Placeholders embedded in the HTML template, beginning with `<3DCityDB>` and ending with `</3DCityDB>` tags, will be replaced in the resulting balloon with the dynamically determined value(s). The HTML balloon templates can also include JavaScript code.

For all concerns, including dynamic content generation, it makes no difference whether the template is taken from the *Balloon_Content* generic attribute or from an external file.

Balloon template format. As previously stated, a balloon template consists of ordinary HTML, which may or may not contain JavaScript code and `<3DCityDB>` placeholders for object-specific content. These placeholders follow several elementary rules.

Rules for simple expressions

- Expressions begin with `<3DCityDB>` and end with `</3DCityDB>`. Expressions are not case-sensitive.
- Expressions are coded in the form `"TABLE/[AGGREGATION FUNCTION] COLUMN [CONDITION]"`. Aggregation function and condition are optional. When present they must be written in square brackets (they belong to the syntax). These expressions represent an alternative coding of a SQL select statement: `SELECT [AGGREGATION FUNCTION] COLUMN FROM TABLE [WHERE condition]`. Tables refer to the underlying 3DCityDB table structure (see chapter 2.3.2 for details).
- Each expression will only return those entries relevant to the city object being currently exported. That means an implicit condition clause somewhat like `"TABLE.CITYOBJECT_ID = CITYOBJECT.ID"` is always considered and does not need to be explicitly written.
- Results will be interpreted and printed in HTML as lists separated by commas. Lists with only one element are the most likely, but not exclusively possible, outcome. When only interested in the first result of a list the aggregation function `FIRST` should be used. Other possible aggregation functions are `LAST`, `MAX`, `MIN`, `AVG`, `SUM` and `COUNT`.
- Conditions can be defined by a simple number (meaning which element from the result list must be taken) or a column name (that must exist in underlying 3DCityDB table structure) a comparison operator and a value. For instance: `[2]` or `[NAME = 'abc']`.
- Invalid results will be silently discarded. Valid results will be delivered exactly as stored in the 3DCityDB tables. Later changes on the returned results - like *substring()* functions - can be achieved by using JavaScript.
- All elements in the result list are always of the same type (the type of the corresponding table column in the underlying 3DCityDB). If different result types must be placed next to each other, then different `<3DCityDB>` expressions must be placed next to each other.

Special keywords in simple expressions

- The balloon template files have several additional placeholders for object-specific content, called `SPECIAL_KEYWORDS`. They refer to data that is not retrieved “as is” in a single step from a table in the 3DCityDB but has to undergo some processing steps (not achievable by simple JavaScript means) in order to calculate the final value before being exported to the balloon. A typical processing step is the transformation of some coordinate list into a CRS different from the one the 3DCityDB is originally set in. The coordinates in the new CRS cannot be included in the balloon with their original values

as read from the database (which was the case with all other expression values so far), but must be transformed prior to their addition to the balloon contents.

- Expressions for special keywords are not case-sensitive. Their syntax is similar to ordinary simple expressions, start and end are marked by `<3DCityDB>` and `</3DCityDB>` tags, the table name must be `SPECIAL_KEYWORDS` (a non-existing table in the 3DCityDB), and the column name must be one of the following:

`CENTROID_WGS84` (coordinates of the object's centroid in WGS84 in the following order: longitude, latitude, altitude)

`CENTROID_WGS84_LAT` (latitude of the object's centroid in WGS84)

`CENTROID_WGS84_LON` (longitude of the object's centroid in WGS84)

`BBOX_WGS84_LAT_MIN` (minimum latitude value of the object's envelope in WGS84)

`BBOX_WGS84_LAT_MAX` (maximum latitude value of the object's envelope in WGS84)

`BBOX_WGS84_LON_MIN` (minimum longitude value of the object's envelope in WGS84)

`BBOX_WGS84_LON_MAX` (maximum longitude value of the object's envelope in WGS84)

`BBOX_WGS84_HEIGHT_MIN` (minimum height value of the object's envelope in WGS84)

`BBOX_WGS84_HEIGHT_MAX` (maximum height value of the object's envelope in WGS84)

`BBOX_WGS84_LAT_LON` (all four latitude and longitude values of the object's envelope in WGS84)

`BBOX_WGS84_LON_LAT` (all four longitude and latitude values of the object's envelope in WGS84)

- No aggregation functions or conditions are allowed for `SPECIAL_KEYWORDS`. If present they will be interpreted as part of the keyword and therefore not recognized.
- The `SPECIAL_KEYWORDS` list is also visible and available in its current state in the updated version of the *Spreadsheet Generator Plugin* (see the following section). The list can be extended in further Importer/Exporter releases.

Examples for simple expressions:

```
<3DCityDB>ADDRESS/STREET</3DCityDB>
```

returns the content of the STREET column on the ADDRESS table for this city object.

```
<3DCityDB>BUILDING/NAME</3DCityDB>
```

returns the content of the NAME column on the BUILDING table for this city object.

```
<3DCityDB>CITYOBJECT_GENERICATTRIB/ATTRNAME</3DCityDB>
```

returns the names of all existing generic attributes for this city object. The names will be separated by commas.

```
<3DCityDB>CITYOBJECT_GENERICATTRIB/REALVAL  
[ATTRNAME = 'H_Trauf_Min']</3DCityDB>
```

returns the value (of the REALVAL column) of the generic attribute with attrname H_Trauf_Min for this city object.

```
<3DCityDB>APPEARANCE/[COUNT]THEME</3DCityDB>
```

returns the number of appearance themes for this city object.

```
<3DCityDB>APPEARANCE/THEME[0]</3DCityDB>
```

returns the first appearance for this city object.

```
<3DCityDB>SPECIAL_KEYWORDS/CENTROID_WGS84_LON</3DCityDB>
```

returns the longitude value of this city object's centroid longitude in WGS84.

<3DCityDB> simple expressions can be used not only for generating text in the balloons, but any valid HTML content, like clickable hyperlinks:

```
<a href="<3DCityDB>EXTERNAL_REFERENCE/URI</3DCityDB>">  
click here for more information</a>
```

returns a hyperlink to the object's external reference,

or embedded images:

```
<img src= "<3DCityDB>CITYOBJECT_GENERICATTRIB/URIVAL  
[ATTRNAME='Illustration']</3DCityDB>" width=400>
```

This last example produces, for instance, in the case of the Pergamon Museum in Berlin:

```

```



Figure 103: Dynamically generated balloon containing an embedded image (image taken from Wikimedia).

Simple expressions are sufficient for most use cases, when only a single value or a list of values from a single column is needed. However, sometimes the user will need to access more than one column at the same time with an unknown amount of results. For these situations (listing of all generic attributes along with their values is one of them) iterative expressions were conceived.

Rules for iterative expressions

- Iterative expressions will adopt the form:

```
<3DCityDB>FOREACH
  TABLE/COLUMN[, COLUMN] [, COLUMN] [...] [, COLUMN] [CONDITION]
</3DCityDB>
[...]
```

 HTML and JavaScript code (column content will be referred to as %1, %2, etc. and follow the columns order in the FOREACH line. %0 is reserved for displaying the current row number)

```
[...]
<3DCityDB>END FOREACH</3DCityDB>
```
- No aggregation functions are allowed for iterative expressions. The amount of columns is free, but they must belong to the same table. Condition is optional. Implicit condition (data must be related to the current city object) applies as for simple expressions.
- FOREACH means truly "for each". No skipping is possible. If skipping at display time is needed it must be achieved by JavaScript means.
- The generated HTML will have as many repetitions of the HTML code between the FOREACH and END FOREACH tags as lines the query result has.
- No inclusion of simple expressions or SPECIAL_KEYWORDS between FOREACH and END FOREACH tags is allowed.
- No nesting of FOREACH statements is allowed.

Examples for iterative expressions:

Listing of generic attributes and their values:

```
<script type="text/javascript">
  function ga_value_as_tooltip(attrname, datatype, strval,
    intval, realval)
  {
    document.write("<span title=\"");
    switch (datatype) {
      case "1": document.write(strval);
        break;
      case "2": document.write(intval);
        break;
      case "3": document.write(realval);
        break;
      default: document.write("unknown");
    };
    document.write("\>" + attrname + "</span>");
  }
</script>
```



```

}

<3DCityDB>FOREACH
  CITYOBJECT_GENERICATTRIB/ATTRNAME, DATATYPE, STRVAL,
  INTVAL, REALVAL</3DCityDB>
  ga_value_as_tooltip("%1", "%2", "%3", "%4", "%5");
<3DCityDB>END FOREACH</3DCityDB>

</script>

```

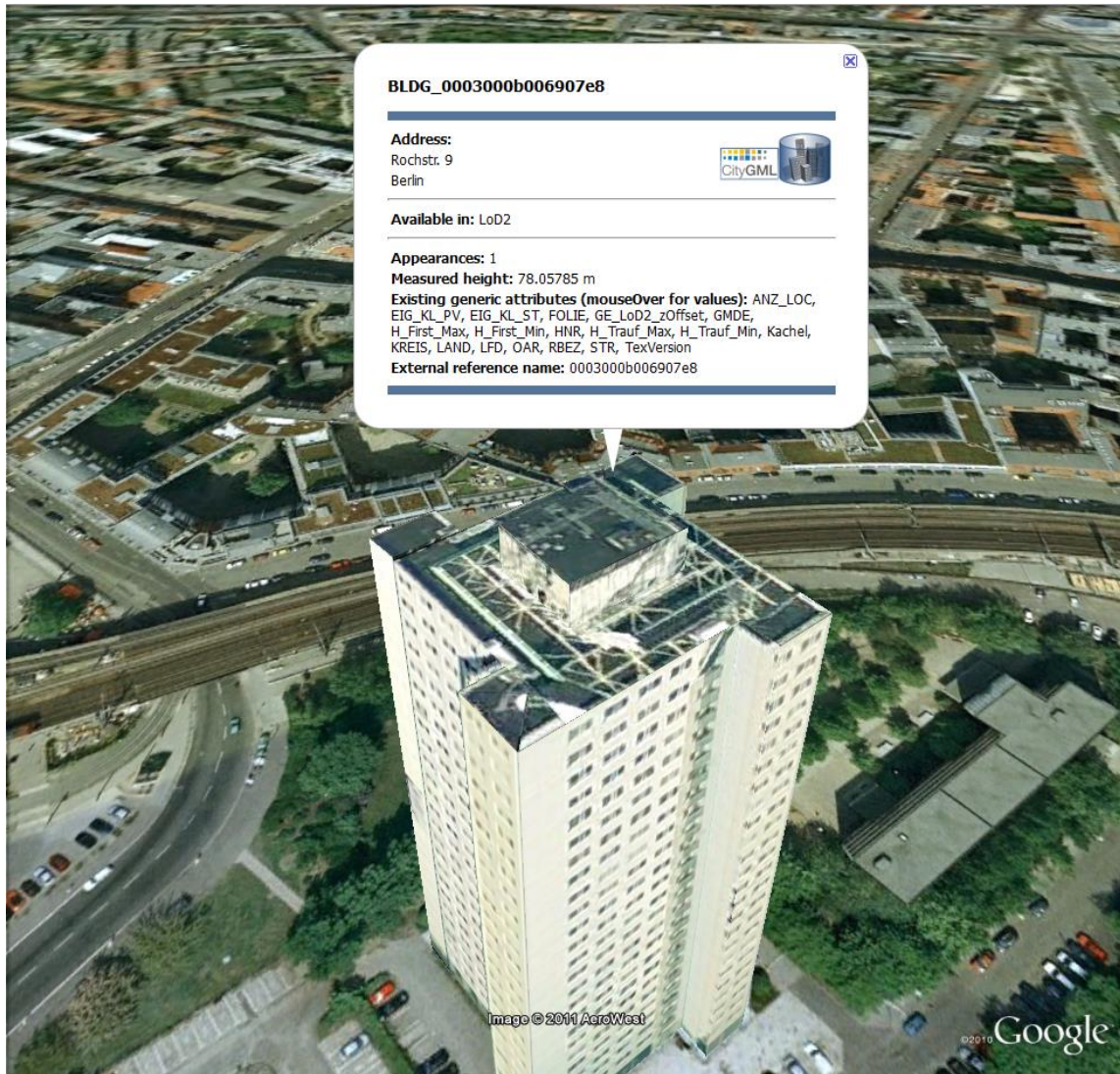


Figure 104: Model placemark with dynamic balloon contents showing the list of generic attributes.

5.6.3.4 Altitude/Terrain Preferences

With reference systems other than WGS84 (the reference system used in Google Earth) in the underlying 3DCityDB, some adjustments on the z coordinate for the exported datasets may be necessary for a perfect display in the Earth browser.

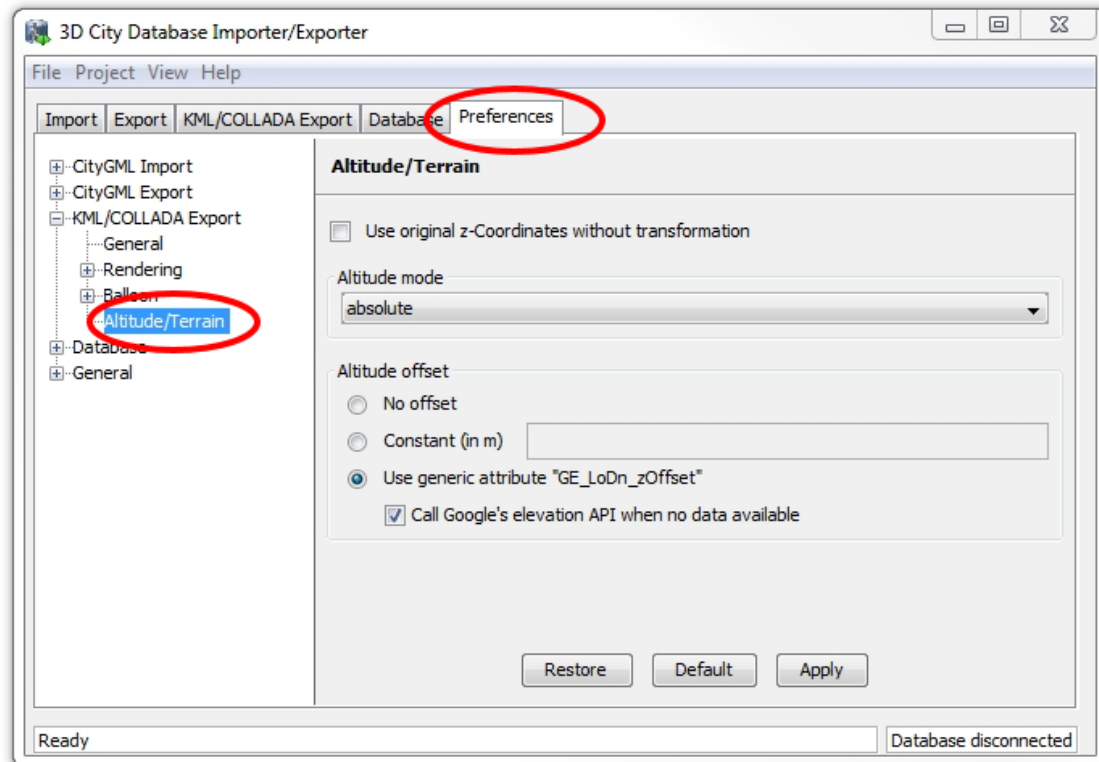


Figure 105: Altitude/Terrain settings.

Use original z-Coordinates without transformation

Depending on the spatial database used, the transformation of the original coordinates to WGS84 will include transformation of the z-coordinates (PostGIS ≥ 2.0 or Oracle $\geq 11g$) or not (Oracle 10g). To make sure only the planimetric (x,y) and not the z-coordinates are transformed this checkbox must be selected. This is useful when the used terrain model is different from Google Earth's and the z-coordinates are known to fit perfectly in that terrain model.

Another positive side-effect of this option is that *GE_LoDn_zOffset* attribute values (explained in the following section) calculated for Oracle 10g keep being valid when imported into PostGIS ≥ 2.0 or Oracle $\geq 11g$. Otherwise, when switching database versions and not making use of this option, *GE_LoDn_zOffset* values must be recalculated again.

GE_LoDn_zOffset attribute values calculated for Oracle 10g are consistent for all KML/COLLADA exports from Oracle 10g. The same applies to PostGIS ≥ 2.0 or Oracle $\geq 11g$. Only cross-usage (calculation in one version, export from the other) creates inconsistencies that can be solved by turning z-coordinate transformation off.

This setting affects the resulting *GE_LoDn_zOffset* if used when a cityobject has none such value yet and is exported in KML/COLLADA for the first time, so it is recommended to remember its status (z-coordinate transformation on or off) for all future exports.

Altitude mode

Allows the user to choose between *relative* (to the ground), interpreting the altitude as a value in meters above the terrain, or *absolute*, interpreting the altitude as an absolute height value in meters according to the vertical reference system (EGM96 geoid in KML).

This means, when *relative* altitude mode is chosen, the z-coordinates of the exports represent the vertical distance from the digital terrain model (DTM) of the Earth browser, which should be 0 for those points on the ground (the building's footprint) and higher for the rest (roof surfaces, for instance). However, z-coordinate values of the city objects stored in a 3DCityDB usually have values bigger than 0, so choosing this altitude mode will result most times in exports hovering over the ground.

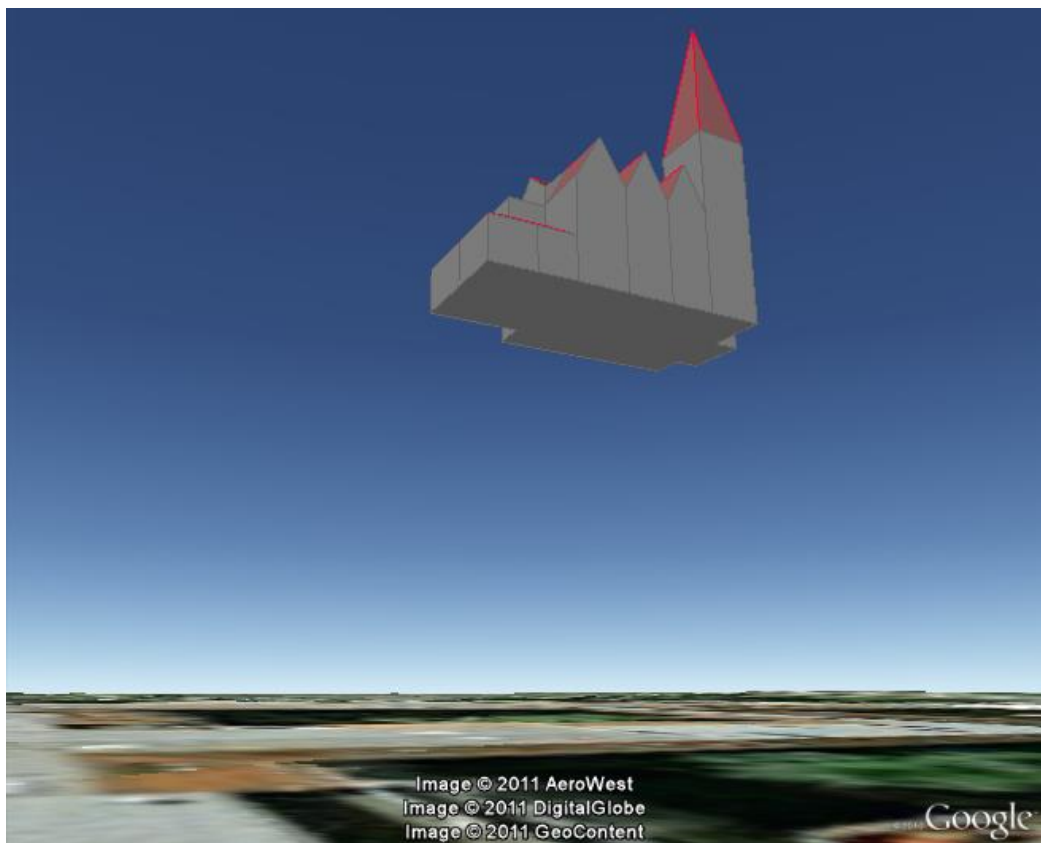


Figure 106: Possible export result with relative altitude mode.

When *absolute* altitude mode is chosen, the z-coordinates of the exports represent the vertical distance from the vertical datum - the ellipsoid or geoid which most closely approximates the Earth curvature, for Google Earth this is the WGS84 EGM96 Geoid, see KML documentation [Wilson 2008] -, regardless of the DTM at that point. This implies, choosing this altitude mode may result in buildings sinking into the ground wherever the DTM indicates there is a hill or hovering over the ground wherever the DTM indicates a dent.

For a proper grounding, a positive or negative offset value can be applied to all z-coordinates of the exports, moving the city objects up and down along the z-axis until they match the ground.

Altitude offset

A value, positive or negative, can be added to the z coordinates of all geometries in one export in order to place them higher or lower over the earth surface. This offset can be 0 for all exported objects (*no offset*), it can be constant for all (*constant*), or it can have an individual value for each object stored in the object's generic attribute *GE_LoDn_zOffset* (where *n* stands for the corresponding level of detail in CityGML sense).

The first two options, *no offset* and *constant*, are appropriate for exports of a single city object, allowing some fine tuning of its position along the z-axis. When exporting regions - via bounding box settings -, the *Use generic attribute "GE_LoDn_zOffset"* option is recommended. Whenever possible (restrictions explained below) settings should be as displayed in Figure 105, including absolute altitude mode. The *GE_LoDn_zOffset* generic attribute value can be automatically calculated by the KML/COLLADA exporter if not available. This calculation uses data returned by Google's Elevation API [Google Elevation API, 2015]. After calculation the value will be stored in the CITYOBJECT_GENERICATTRIB table of the 3DCityDB for future use.

Since city objects may have different geometries for different LoDs, the anchoring points and their elevation values may also differ for each LoD. This explains the need for having *GE_LoD1_zOffset*, *GE_LoD2_zOffset*, etc. generic attributes for one single object.

The algorithm used to calculate the individual zOffset for an object iterates over the points with the lowest z-coordinate in the object, calling Google's elevation API in order to get their elevation. The point with the lowest elevation value will be chosen for anchoring the object to the ground. The zOffset value results from subtracting the point's z-coordinate from the point's elevation value.

When calling Google's elevation API for calculating the zOffset of an object a message is shown: "Getting zOffset from Google's elevation service for BLDG_0003000e008c4dc4".

Google's elevation API imposes strong usage restrictions: non-premium users can issue a maximum of 2,500 requests per day. This limit may be reached fast when exporting areas where no city objects have *GE_LoDn_zOffset* values assigned. When the daily usage limit is reached a warning message is shown: "Elevation service returned OVER_QUERY_LIMIT". The usage limit is bound to the caller's IP address. It is advisable to use several different computers (or IP addresses) when filling the 3DCityDB with *GE_LoDn_zOffset* values (or become a premium user).

A second usage restriction allows for no more than 10 requests per second. The Import/Export tool takes care of not exceeding this limit by pausing between requests when required. That will slow down KML/COLLADA exports when done for the first time. Subsequent exports will be

faster since the *GE_LoDn_zOffset* attribute value is already in the 3DCityDB and does not have to be calculated again.

Saving the building's height offset in the form of a generic attribute ensures this information will be present in every export in CityGML format (and therefore at every re-import) and can thus be transported across databases. Please note, that not the DTM height value of Google Earth will be stored but the difference of the individual building's minimum z value and the value reported by the Google Elevation Service. Following this approach further usage restrictions of the Google Elevation Service are avoided.

In some unusual cases, even after automatic calculation of the *GE_LoDn_zOffset* value the object may still not be perfectly grounded to the Earth surface for a number of reasons; e.g. wrong height data of the model, or low resolution of the DTM at that area. In those cases a manual adjustment of the value in the 3DCityDB is needed. After the content of *GE_LoDn_zOffset* has been fine tuned to a proper value it should be persistently stored in the database.

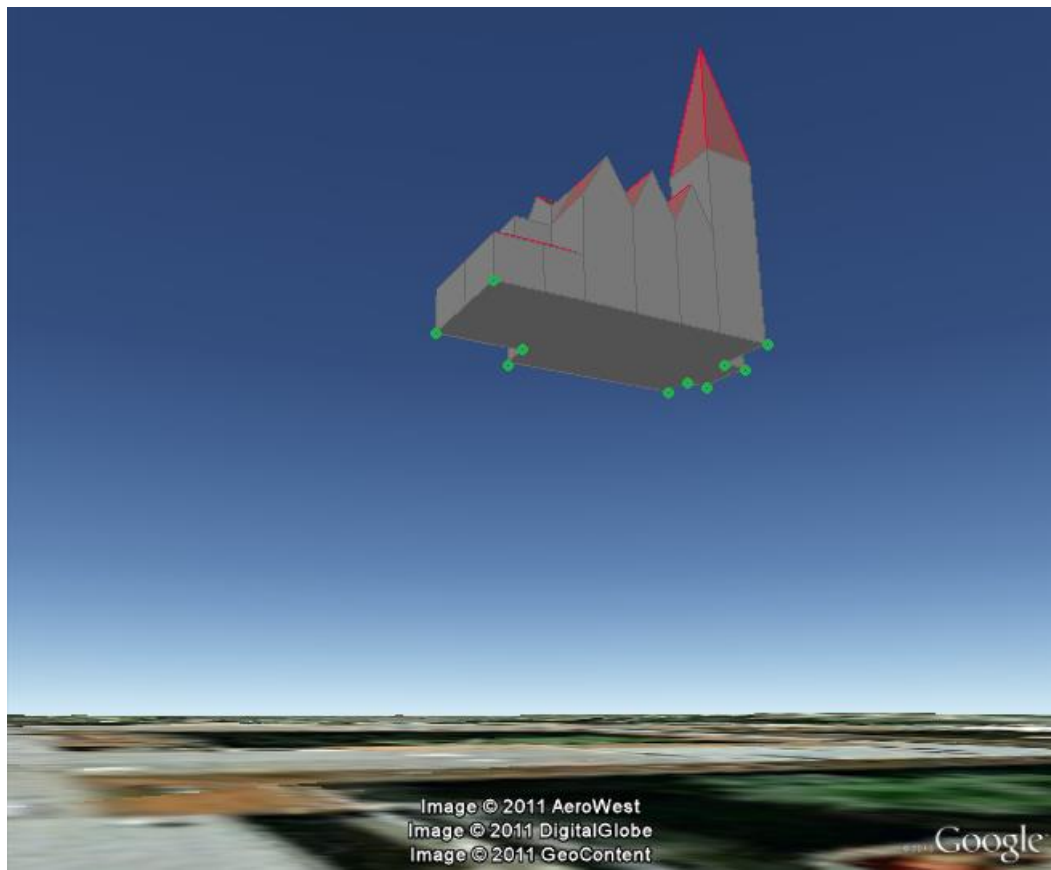


Figure 107: Points sent to Google's Elevation API for calculation of the zOffset.

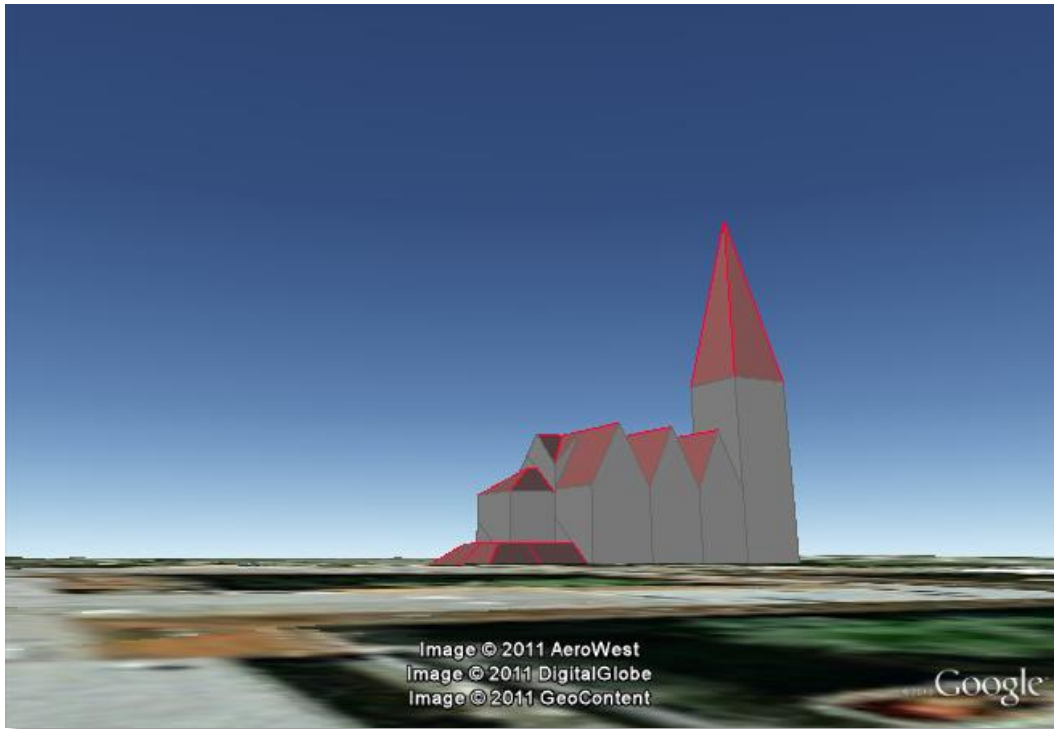


Figure 108: Export with *absolute* altitude mode and *no offset*.

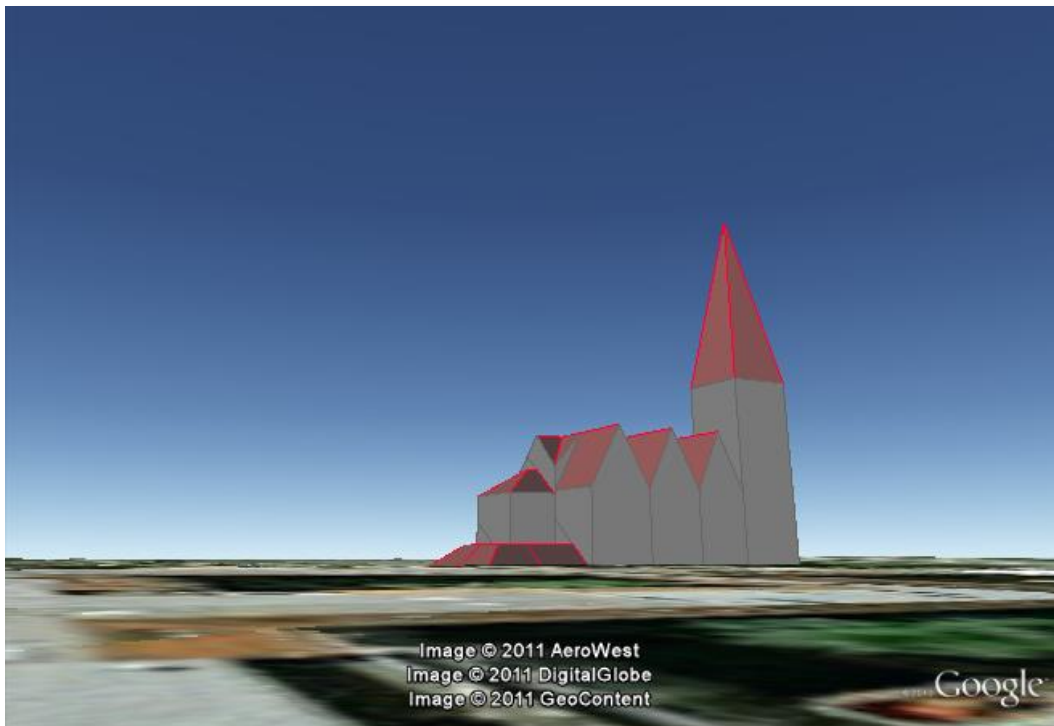


Figure 109: Export with *absolute* altitude mode and use of *GE_LoDn_zOffset*.

5.6.3.5 General setting recommendations

Depending on the quality and complexity of the 3DCityDB data, export results may vary greatly in aesthetic and loading performance. Experimenting will be required in most cases for a fine tuning of the export parameters. However, some rules apply for almost all cases:

- kmz format use is recommended when the files will be accessed over a network.
- Visibility values for the different display forms should be increased in steps of around one third of the tile side length.
- Visibility from 0 pixels (always visible) should be avoided, especially for large or complex exports, because otherwise the Earth browser will immediately load all data at once since it all must be visible.
- Tile side length (whether tiling is *automatic* or *manual*) should be chosen so that the resulting tile files are smaller than 10MB. When single files are bigger than that Google Earth gets unresponsive. For densely urbanized areas, where many placemarks are crimped together a tile side length value between 50 and 100m should be used.
- When not exporting in the *COLLADA* display form, files will seldom reach this 10MB size, but Google Earth will also become unresponsive if the file loaded contains a lot of polygons, so do not use too large tiles for *footprint*, *extruded* or *geometry* exports even if the resulting files are comparatively small.
- Do not choose too small tile sizes, many of them may become visible at the same time and render the tiling advantage useless.
- Using texture atlas generation when producing *COLLADA* display form exports always results in faster model loading times.
- From all texture atlas generating algorithms, *BASIC* is the fastest (shortest generation time), *TPIM* the most efficient (highest used area/total atlas size ratio).
- Texture images can often be scaled down to 0.2 - 0.5 without noticeable quality loss. This depends, of course, on the quality of the original textures.
- Highlighting puts the same polygons twice in the resulting export files, one for the buildings themselves, one for their highlighting. This has a negative impact on the viewing performance. The more complex the buildings are the worse the impact. When highlighting is enabled for exports based on a CityGML LoD3 or higher Google Earth may become quite slow.
- Balloon generation is slightly more efficient when a single template file is applied for all exported objects.
- Optimal altitude/terrain settings for a proper grounding of the exports are as shown in Figure 105: absolute altitude mode, use of generic attribute *GE_LoDn_zOffset* and call Google's elevation API when no data is available.
- When the Google's elevation API daily quota limit is reached you can continue the export on another computer, or you can change your IP address (or become a Google premium user). Repetitive running of the KML/COLLADA export may be required over several days until error message "OVER_QUERY_LIMIT" no longer appears.

5.6.4 Management of user-defined coordinate reference systems

When setting up an instance of the 3D City Database, a coordinate reference system (CRS) has to be chosen for the entire database (cf. chapter 3.3). This CRS is used as default reference system for all spatial objects that are created and stored in the database instance (except implicit geometries) as well as for building spatial indexes and performing spatial functions.

At many places, the Importer/Exporter allows for providing coordinate values associated with a different CRS though, e.g. when defining *spatial bounding box filters* for CityGML imports and exports and KML/COLLADA exports, or when defining a *target CRS* into which coordinate values shall be converted during CityGML exports (see the documentation of the corresponding operations). In order to add and manage additional reference systems, the Importer/Exporter provides a corresponding dialog on the Preferences (*Reference systems* subnode of the *Database* preferences node) tab as shown below.

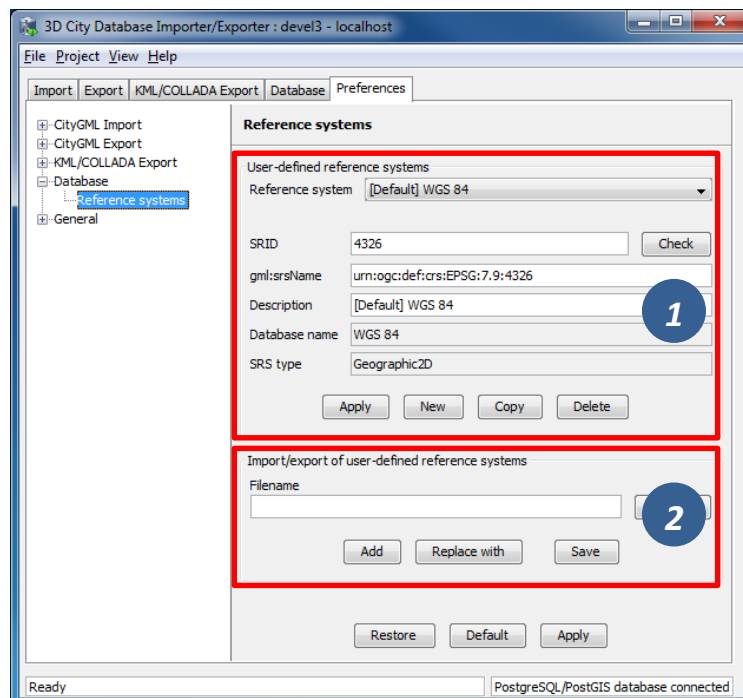


Figure 110: Database preferences – Reference systems.

On top of the preferences page [1], a drop-down list allows for choosing a CRS for display and editing from the list of user-defined CRSs. This list contains at minimum one predefined entry called *Same as in database* which represents the internal CRS of the 3D City Database instance. This entry will always show the SRID and CRS URN encoding of the currently connected database instance. Since the internal CRS shall not be changed after database setup using the Importer/Exporter, the fields of the *Same as in database* entry cannot be edited.

A new user-defined CRS can be added to this list after clicking the *New* button. Please provide the database-internal SRID in the corresponding *SRID* input field of the user dialog and enter the URN encoding of the CRS into the *gml:srsName* input field (optional). A short, meaningful textual description of the CRS has to be provided in the *Description* field. This description is used as value for the drop-down on top of the dialog, but also for similar CRS drop-down lists

on further tabs of the Importer/Exporter. The new CRS is added to the list of user-defined CRSs upon clicking the *Apply* button. The following screenshot provides an example.

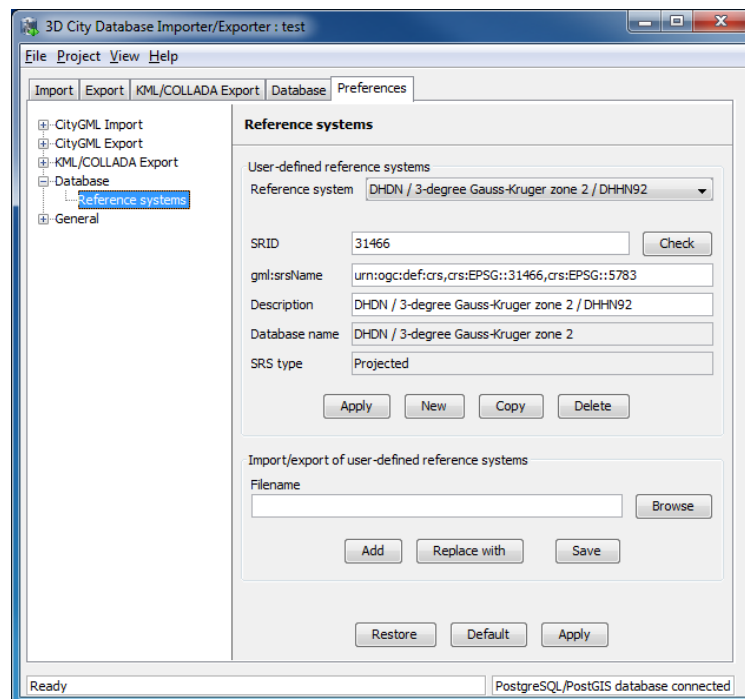


Figure 111: Adding a new CRS to the list of user-defined CRSs.

The *Copy* button allows for adding a further CRS by copying and editing the information of an already existing user-defined CRS. The currently selected CRS is deleted from the list by clicking the *Delete* button. The *Check* button next to the *SRID* input field facilitates to verify whether the provided *SRID* is supported by the currently connected 3D City Database instance. After a successful check, the non-editable fields *Database name* and *SRS type* will be filled with the corresponding information collected from the currently connected 3D City Database instance. If the Importer/Exporter is not connected to a database instance, the *Check* button is disabled.

The result of the *SRID* verification may vary between different 3D City Database instances since 1) the list of predefined spatial reference systems differs between different database systems and versions and 2) both Oracle and PostgreSQL/PostGIS support the definition of user-defined spatial reference systems on the database side (please check the respective database documentation for guidance).

Note: In order to add a user-defined CRS to the Importer/Exporter that is not supported by the underlying Oracle or PostgreSQL/PostGIS database, you need to first register this CRS in your database. As soon as the CRS is available from the database, it can be added to the list of user-defined CRSs in the Importer/Exporter.

The list of user-defined CRSs is automatically stored in the config file of the Importer/Exporter and loaded upon application start. It can additionally be exported into an extra file (see [2] in Figure 110). This allows for easily sharing user-defined CRSs between different installations of the Importer/Exporter. Please provide a valid filename in the corresponding input field *Filename* (use the *Browse* button to open a file selection dialog) and click on *Save*. There are

two more options for importing such an external list of CRSs: 1) the CRSs listed in the external file can be added to the current list of CRSs (*Add* button) or 2) the external list can be used to replace the current list (*Replace with* button).

The Importer/Exporter is shipped with a number of predefined CRSs organized in subfolders below `templates/CoordinateReferenceSystems` in the installation folder. Each CRS definition is stored in its own file and, thus, can be easily imported and added to the list of user-defined CRSs. Note that the URN encoding of the predefined CRSs generally lacks a height reference system. The height reference therefore has to be added before using this CRS as target reference system for CityGML exports (cf. chapter 5.4 for more details).

5.6.5 General preferences

In addition to the preference settings that influence the behavior of a particular import or export operation (cf. previous sections), the `General` node on the `Preferences` tab offers application-wide settings.

5.6.5.1 Cache

Both during CityGML imports at exports, the Importer/Exporter has to keep track of various temporary information. For instance, when resolving XLinks, the `gml:id` values as well as additional information about the related features and geometries must be available. Since the Importer/Exporter is designed to be able to process arbitrarily large CityGML input files, keeping this information in main memory only is not a promising strategy. For this reason, the information is written to *temporary tables* in the database as soon as user-defined memory limits are reached.

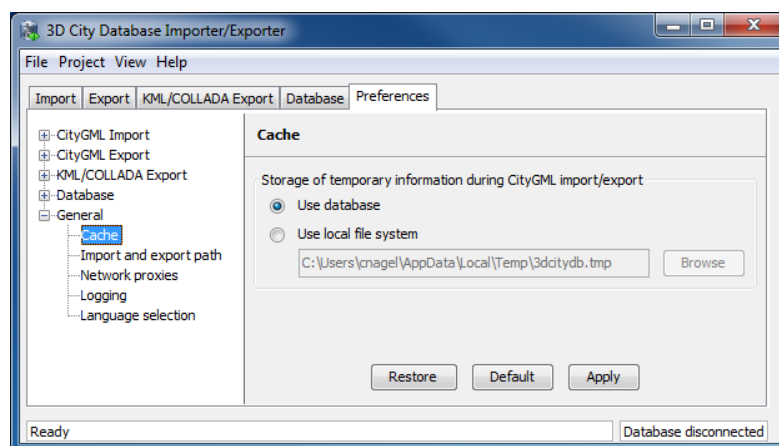


Figure 112: General preferences – Cache.

Per default, temporary tables are created in the *3D City Database instance* itself. The tables are populated during the import and export operation and are automatically dropped after the operation has finished. Alternatively, the user can choose to store the temporary information in the *local file system* instead. An absolute path where to create the file-based storage has to be provided. Either type the location manually into the input field, or use the *Browse* button to open a file selection dialog. A subfolder of the local *temp folder* of the operating system user running the Importer/Exporter is proposed as default location (depends on the operating system

in use). Like with temporary database tables, the file-based storage is automatically removed after the operation has finished.

Some reasons for using a file-based storage are:

- The 3D City Database instance is kept clean from any additional (temporary) table.
- If the Importer/Exporter runs on a different machine than the 3D City Database instance, sending temporary information over the network might be slow. In such cases, using a local storage might help to increase performance.

5.6.5.2 Import and export path

This preference dialog allows for setting a default path for import and export operations.

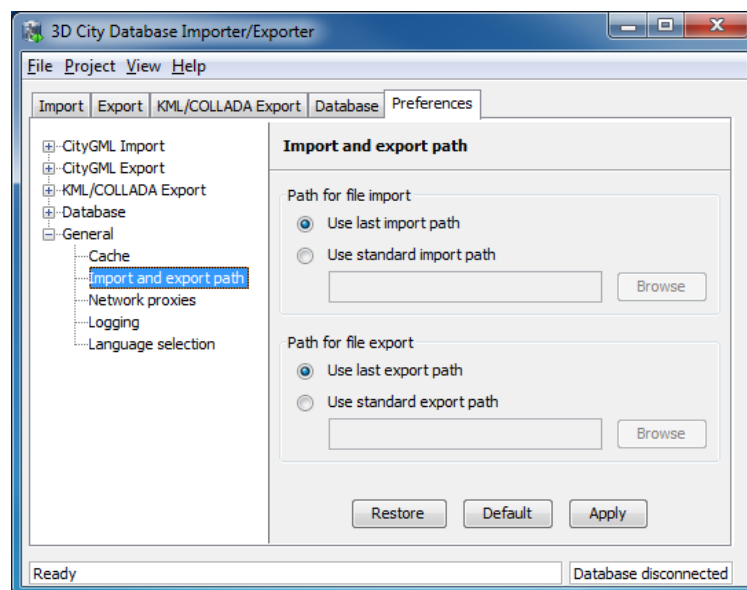


Figure 113: General preferences – Import and export path.

Simply choose between the last used import/export path (default), or browse for a specific folder in your local file system. The selected folder will then be used as default path in all dialogs that require an input/output file.

5.6.5.3 Network proxies

Some of the functionalities offered by the Importer/Exporter require internet access. This applies, for instance, to the XML validation when accessing XML Schema documents on the web, to the map window for the graphical selection of bounding boxes (uses *OpenStreetMap* data), or to the automated calculation of height offsets during KML/COLLADA exports (based on the *Google elevation service*).

Most computers in corporate environments have no direct internet access but must use a proxy server. The preference dialog shown below let you configure network proxies.

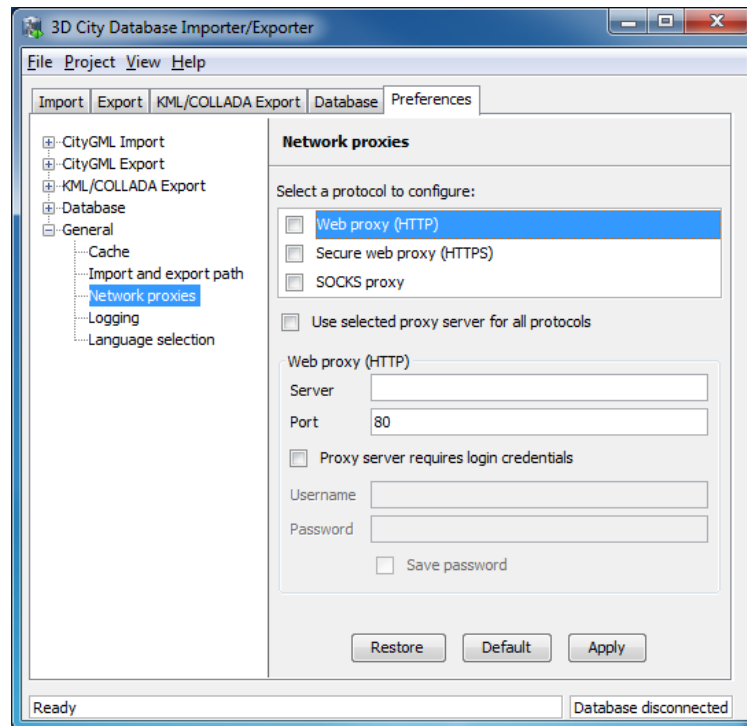


Figure 114: General preferences – Network proxies.

The Importer/Exporter supports *Web* (*HTTP*), *Secure web* (*HTTPS*) and *SOCKS* proxies. Usually, configuring a *Web proxy* (*HTTP*) is enough for most tasks, like those mentioned above. However, more sophisticated use cases, like uploading cloud documents via an Importer/Exporter extension plugin (cf. chapter 6.2) may require *Secure web proxy* (*HTTPS*) support. *SOCKS proxy* support should currently only be needed when the Importer/Exporter and the database system running the 3D City Database reside in different networks.

Whenever one of the protocols to be handled by a proxy is selected in the choice list at the top of the dialog, the corresponding settings have to be provided in the fields below: *Server*, *Port*, and if the proxy requires login credentials *Username* and *Password*. Default *Port* values for each protocol are automatically filled in (*HTTP*: 80; *HTTPS*: 443; *SOCKS*: 1080) and only need to be changed if required.

It is also possible to define one single proxy for all protocols by simply selecting the corresponding checkbox under the protocol list. Just make sure the proxy server supports all protocols and that they can all be routed through the given *Port*.

Proxies are only used if the checkbox next to the protocol type is enabled. Otherwise, the proxy configuration will be stored but remains inactive. When the proxy for a given protocol is enabled, every outgoing connection by the Importer/Exporter that uses the protocol will be routed through this proxy.

In case the computer running the Importer/Exporter is directly connected to the internet no proxies need to be configured.

5.6.5.4 Logging

The Importer/Exporter logs information about events such as activities or failures, for instance during database imports and exports. Each log entry consists of a timestamp when the event occurred, a log level indicating the severity of the event and a human-readable message text. Log messages are always printed to the *console window* and may additionally be forwarded to a log file on your local computer. The Logging preference dialog is shown below.

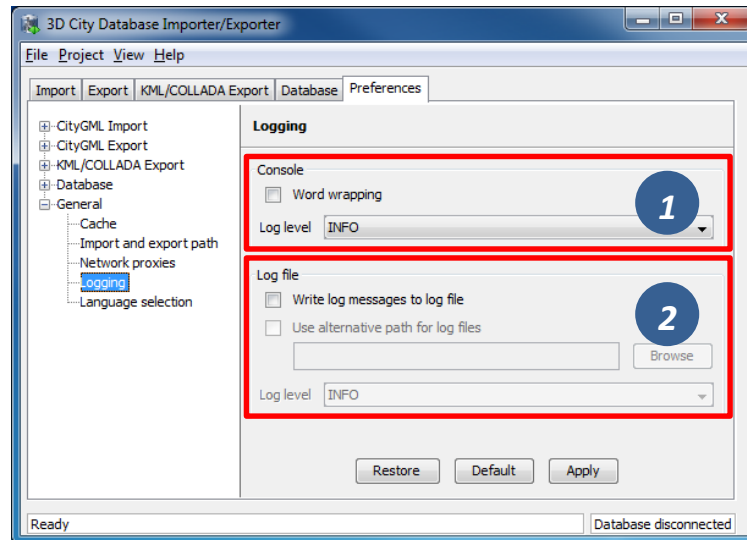


Figure 115: General preferences – Logging.

The following four log levels are distinguished (from highest to lowest severity):

- *ERROR* An error has occurred (usually an exception). This comprises internal and unexpected failures. Moreover, invalid XML content of CityGML instance documents is reported via this log level. Fatal errors will cause the running operation to abort.
- *WARN* An unusual condition has been detected. The operation in progress continues to work but the user should check the warning and take appropriate actions.
- *INFO* An interesting piece of information about the current operation that helps to give context to the log, often when processes are starting or stopping.
- *DEBUG* Additional messages reporting the internal state of the application.

The log level for messages printed to the console window can be chosen from a drop-down list in the *Console* dialog [1]. The log will include all events of the indicated severity as well as events of greater severity (default: *INFO*). *Word wrapping* can be optionally enabled for long message texts that otherwise exceed the width of the console window.

Note: The log output in the *console window* is truncated after 10,000 log messages in order to prevent high main memory consumption.

If log messages shall additionally be stored in a log file, simply activate the option *Write messages to log file*. The log file is named `log_3dcitydb_impexp_<date>.log` per

default, where <date> is replaced with the current date at program startup. The Importer/Exporter creates the log file if it does not exist. Otherwise, log messages are appended to the existing log file. The user can choose a location where to store the log file by enabling the option *Use alternative path for log files* and by providing a corresponding path [2]. Either enter the path manually or click on *Browse* to open a file selection dialog. The log level can be chosen independent from the console window through the corresponding drop-down list [2] (default: *INFO*).

Note: Log files are per default stored in the *home directory* of the *operating system user* running the Importer/Exporter. Precisely, you will find the log files in the subfolder `3dcitydb/importer-exporter-3.0/log`. However, the location of the home directory differs for different operating systems. Using environment variables, the location can be identified dynamically:

- %HOMEDRIVE%%HOMEPATH%\3dcitydb\importer-exporter-3.0\log (Windows 7 and higher)
- \$HOME/3dcitydb/importer-exporter-3.0/log (UNIX/Linux, Mac OS families)

5.6.5.5 Language selection

The Importer/Exporter GUI has support for different languages. Use the Language selection preference dialog shown below to pick your favourite language.

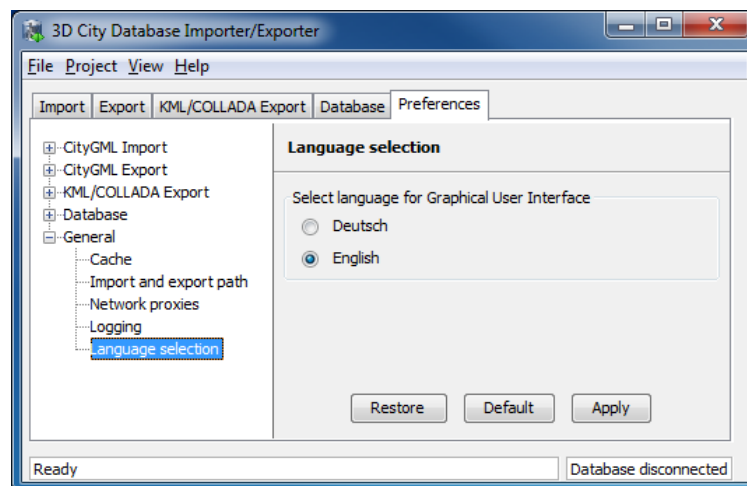


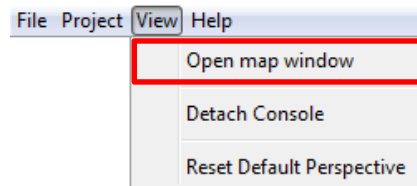
Figure 116: General preferences – Language selection.


5.7 Map window for bounding box selections

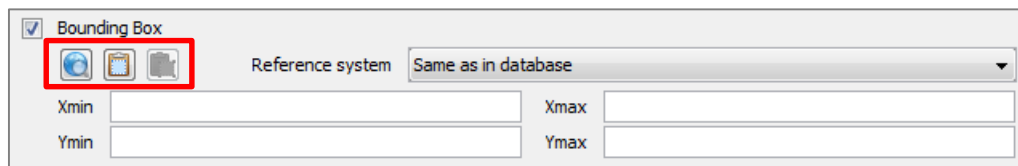
The Importer/Exporter GUI offers a 2D map window that allows the user to display the overall bounding box calculated from the city model content stored in a given 3D City Database instance and to graphically select a bounding box filter for data imports and exports.

There are two ways to open the map windows:

1. Choose the entry View → Open map window from the menu bar at the top of the application window.



2. Click the map button  on the bounding box dialog available on the Import, Export, KML/COLLADA export and Database tabs of the operations window.



The 2D map is rendered in a separate application window shown below.

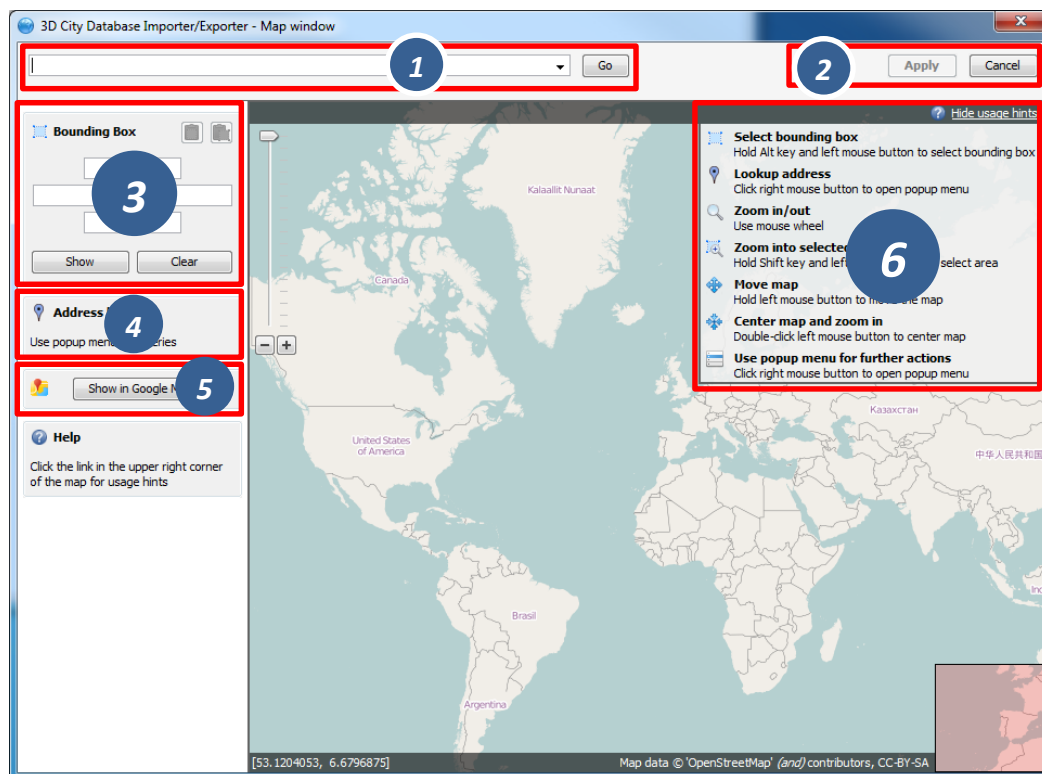




Figure 117: 2D map window for bounding box selections.

The map content is provided by the *OpenStreetMap* service and comes without usage limits. Make sure your computer has internet access to load the map. This might require setting up *network proxies* (see chapter 5.6.5.3). Please consult your network administrator.

The map offers default mouse controls for panning and zooming. For convenience, a geocoding service is included in the map window [1]. Simply type in an address or a geo location (given by geographic lat/lon coordinates separated by a comma) and click the *Go* button. The map will automatically zoom to the first match. Further matches are available from the drop-down list [1]. The geocoding service uses the free *Google Geocoding API* and is therefore subject to Google's terms of use. In general, it is limited to 2,500 requests per day and IP address. To display the result of the geocoding query on Google Maps in your default internet browser, simply click the *Show in Google Maps* button [5].

A list of usage hints is available at the right top of the map window [6]. Please click on the *Show usage hints* link to display this list. The map controls are also described in the following.

- *Select bounding box*: Move the mouse while pressing the ALT key and the left mouse button to select a bounding box. The bounding box is displayed in a light magenta color. Once the left mouse button is released, the coordinates of the bounding box are automatically filled in the Bounding Box dialog on left of the map [3]. If you have opened the map window from a bounding box filter dialog, then clicking the *Apply* button on the upper right corner of the window [5] closes the map window and carries the bounding box values to the filter dialog. In addition, the values are copied to the clipboard.
- *Lookup address*: Right-click on the map to bring up a context menu for the geo location at the mouse pointer. From the context menu, choose *Lookup address here*. This will trigger a reverse geocoding query through the *Google Geocoding API*. The resulting address will be displayed on the left of the window [4]. The  icon denotes which location on the map is associated with the address, whereas the  icon shows where you clicked on the map (see Figure 118).
- *Zoom in/out*: Use the mouse wheel or the context menu (right-click).
- *Zoom into selected area*: Move the mouse while pressing the SHIFT key and the left mouse button to select an area. The selected area is displayed in a light grey color. Once the left mouse button is released, the map zooms into the selected area. If the maximum zoom level is reached this action has no further effect.
- *Move map*: Keep the left mouse button pressed to move the map.
- *Center map and zoom in*: Double click the left mouse button to center the map at that position and to increase the current zoom level by one step.

- *Use popup menu for further actions:* Right-click on the map to bring up a context menu offering additional functions such as *Zoom in*, *Zoom out*, *Center map here* and *Lookup address here* (see above). The *Get map bounds* function is equivalent to selecting the visible map content as bounding box. Thus, the map will be shown in light magenta and the map bounds are transferred to the Bounding Box dialog on the left [3].

To close the map, simply click the *Cancel* button in the upper right corner [2].

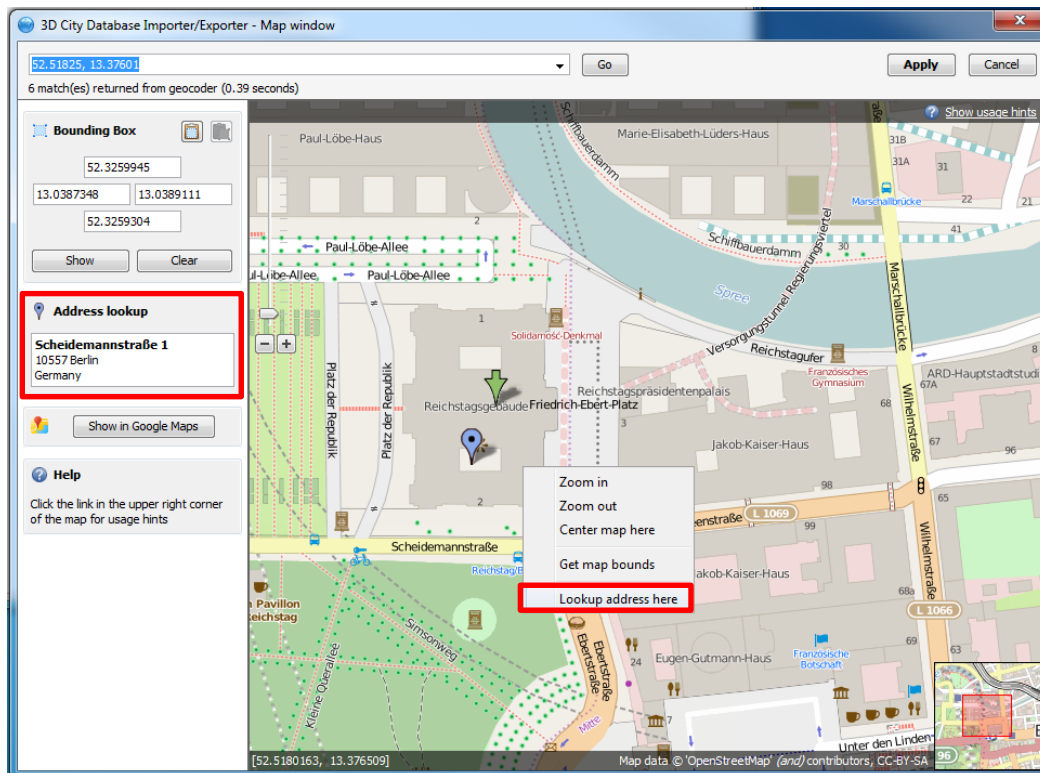


Figure 118: Address lookup in the map window.

The coordinates in the map window and of the selected bounding box are always given in WGS 84 regardless of the coordinate reference system of the 3D City Database instance.

When opening the map window from a bounding box dialog that already contains coordinate values (e.g., from a filter dialog on the Import, Export or KML/COLLADA export tabs or after having calculated the entire area of the database content on the Database tab), the map window will automatically display this bounding box. If the coordinate values of the provided bounding box are not in WGS 84, a transformation to WGS 84 is required. Since the Importer/Exporter uses functionality of the underlying spatial database system for coordinate transformations, a connection to the database must have been established beforehand. In case there is no active database connection, the following pop-up window asks the user for permission to connect to the database.

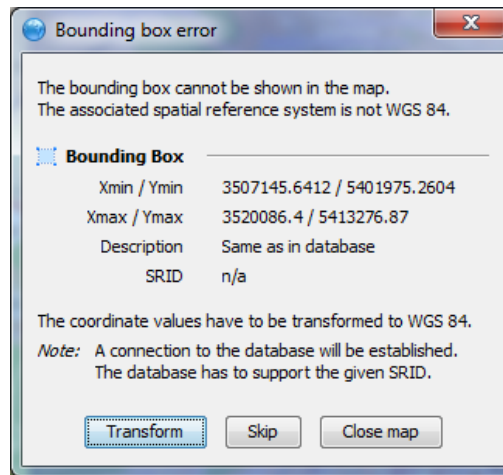




Figure 119: Asking for permission before connecting to a database for coordinate transformation.

The *Apply* button on the upper right corner of the map window [2] is a shortcut for copying the coordinate values to the clipboard and pasting them in the bounding box fields of the calling tab on the operations window. Furthermore, coordinate values can now be easily copied from one tab to another by simply clicking on the copy button  in one of them, say *Import* tab, with filled *bounding box* values, changing to another, say *KML/COLLADA Export* tab and clicking on the  button there. Previously existing values in the bounding box fields of the *KML/COLLADA Export* tab (if any) will be overwritten.

5.8 Using the command line interface (CLI)

In addition to the graphical user interface, the Importer/Exporter also offers a command line interface (CLI). The CLI allows a user to run the Importer/Exporter from the command line (or a shell script) and to easily embed it in batch processing workflows and third party applications.

To use the CLI, you first need to start a shell environment offered by the operating system of your choice. The general command to run the Importer/Exporter from a shell environment (or a shell script) is shown below.

```
java -jar lib/3dcitydb-impexp.jar [-options]
```

This command consists of two parts. The first part executes the *Java Virtual Machine* (JVM) through the `java` command. The `-jar` argument of the JVM is used to denote the path to the Importer/Exporter JAR file `3dcitydb-impexp.jar` to be executed. After the JAR filename, you have to provide additional program arguments to trigger a specific operation of the Importer/Exporter.

Note: The above command assumes that you have first changed directory to the directory where the Importer/Exporter is installed. Otherwise, you have to provide the full path to the `3dcitydb-impexp.jar` file.

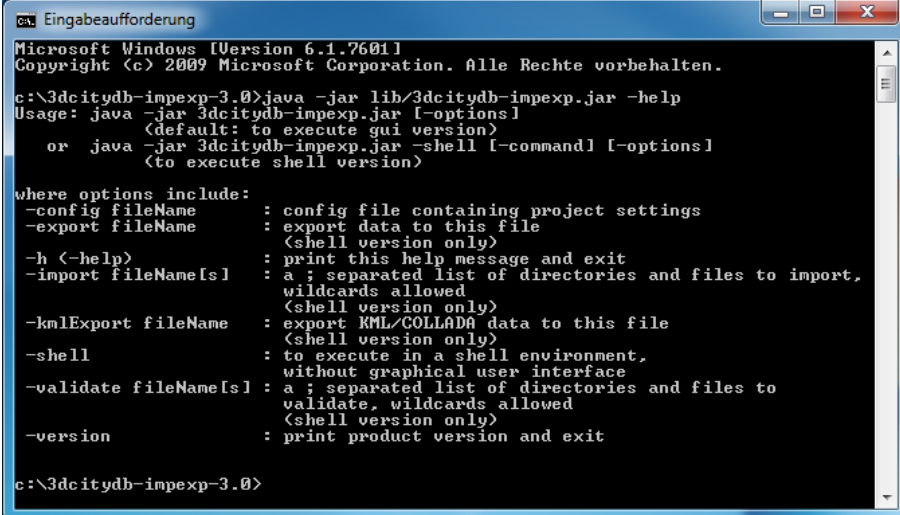
You may add any further JVM arguments to the above command that you think are required in your environment. *It is recommended* to at least increase the *maximum available heap space*

using the `-Xmx` argument. For instance, use `java -Xmx 1024m` to allow 1GB of your main memory to be used by the Importer/Exporter.

In order to get a list of program arguments offered by the Importer/Exporter, use the `-help` flag and issue the following command:

```
java -jar lib/3dcitydb-impexp.jar -help
```

This will produce an output like shown below.



```

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Alle Rechte vorbehalten.

c:\3dcitydb-impexp-3.0>java -jar lib/3dcitydb-impexp.jar -help
Usage: java -jar 3dcitydb-impexp.jar [-options]
       or java -jar 3dcitydb-impexp.jar -shell [-command] [-options]
       (to execute shell version)

where options include:
  -config fileName      : config file containing project settings
  -export fileName      : export data to this file
                        <shell version only>
  -h <-help>           : print this help message and exit
  -import fileName[s]  : a ; separated list of directories and files to import,
                        wildcards allowed
                        <shell version only>
  -kmlExport fileName  : export KML/COLLADA data to this file
                        <shell version only>
  -shell               : to execute in a shell environment,
                        without graphical user interface
  -validate fileName[s]: a ; separated list of directories and files to
                        validate, wildcards allowed
                        <shell version only>
  -version             : print product version and exit

c:\3dcitydb-impexp-3.0>

```

Figure 120: Help text of the command line interface.

The available program arguments are:

- | | |
|------------------|---|
| <i>-shell</i> | This argument is mandatory to start the shell version of the Importer/Exporter. If this argument is not provided, then the GUI version is launched per default. |
| <i>-config</i> | Provides the path and filename of the config file to be used. If this argument is omitted, the config file in the default path is used instead. Using environment variables, the default path can be identified dynamically (cf. chapter 5.1): <ul style="list-style-type: none"> • %HOMEDRIVE%%HOMEPATH%\3dcitydb\importer-exporter-3.0\config (Windows 7 and higher) • \$HOME/3dcitydb/importer-exporter-3.0/config (UNIX/Linux, Mac OS families) |
| <i>-import</i> | Triggers a CityGML import process. Provide a list of one or more input files separated by semicolons (;) in addition. The list may also contain folders. A folder and all its nested subfolders are recursively scanned for CityGML input files. |
| <i>-validate</i> | Triggers a XML Schema validation on the provided list of input files (see import argument). |

<code>-export</code>	Triggers a CityGML export process. Provide the path and name of the output file.
<code>-kmlExport</code>	Triggers a KML/COLLADA export process. Provide the path and name of the output file.

The full range of preferences and settings affecting the different import and export operations of the Importer/Exporter **are not offered** as separate program arguments. Instead, it is assumed that the config file (either the default one or the one provided through the `-config` argument) contains all the settings that should be used in a specific operation (e.g., the database connection details, filter settings for imports and exports, etc.). The config file is encoded as XML and hence can be edited by a user manually. However, the recommended way to provide valid settings is as follows:

1. Run the Importer/Exporter with the graphical user interface (GUI).
2. Make all your settings in the GUI.
3. Save your settings to a local config file via the `Project → Save Project As...` dialog from the main menu bar.
4. Feed this config file to the command line interface using the `-config` argument.

Note: You can also create a config file programmatically in Java. For this purpose, you can build a JAR file containing all the necessary classes for reading and writing a config file. You need to obtain the source code of the Importer/Exporter, which provides an ANT build script. Use the `generate_standalone_config` target to build the JAR file. Once you have the JAR file on your classpath, use the class `ConfigUtil.java` as starting point.

6 Importer / Exporter plugins

6.1 Introduction to the plugin architecture

The Importer/Exporter offers a plugin architecture that supports the modular development and deployment of additional functionalities for interacting with the 3D City Database or external datasets. For instance, plugins may enable loading or extracting 3D city model content using data formats other than CityGML or KML/COLLADA. Plugins are self-contained extensions in that one plugin cannot extend the functionality of another plugin. Therefore, plugins can be added separately to the Importer/Exporter without interdependencies.

A plugin may extend the GUI of the Importer/Exporter by providing its own user dialog that will be rendered in a separate tab on the operations window. In addition, a plugin may add new entries to the main menu bar and the preferences dialog. In order to remember the preference settings at program startup, a plugin can choose to serialize the settings to the main config file or a plugin-specific config file. Please refer to the plugin documentation of your vendor for more information.

Plugin installation is simple. Just get the plugin from your plugin vendor and put all plugin files into the `plugins` subfolder of the Importer/Exporter installation directory. In order to keep multiple plugins independent from each other, it is recommended to create a separate subfolder below `plugins` for each plugin. When running the Importer/Exporter, the installed plugins are automatically detected and loaded with the application.

Currently, a *Spreadsheet Generator* plugin is shipped with the Importer/Exporter (cf. Figure 121). This free and open-source plugin allows general purpose exports of 3D city model contents as spreadsheets, either to a local `.csv` file with user-defined formatting or directly to an online spreadsheet hosted in the cloud. A comprehensive documentation of the *Spreadsheet Generator* plugin is provided in the following chapter 6.2.

Note: Previous versions of the Importer/Exporter additionally offered a *Matching/Merging* plugin. The development of this plugin has been discontinued though.

You can also develop your own plugins. For this purpose, the Importer/Exporter comes with a *Plugin API* that is available as separate JAR file. During setup, you can choose to install this Plugin API, which will copy the JAR file `3dcitydb-impexp-plugin-api.jar` to its own `plugin-api` subfolder below the Importer/Exporter installation directory. The JAR file is accompanied with a `README.txt` file, license information, and a Javadoc documentation. Simply put the JAR file on your classpath to start plugin development. A comprehensive *Plugin API* guide will be offered on the www.3dcitydb.org website in the near future. Moreover, the source code of the *Spreadsheet Generator* plugin can be used as template for your own developments.

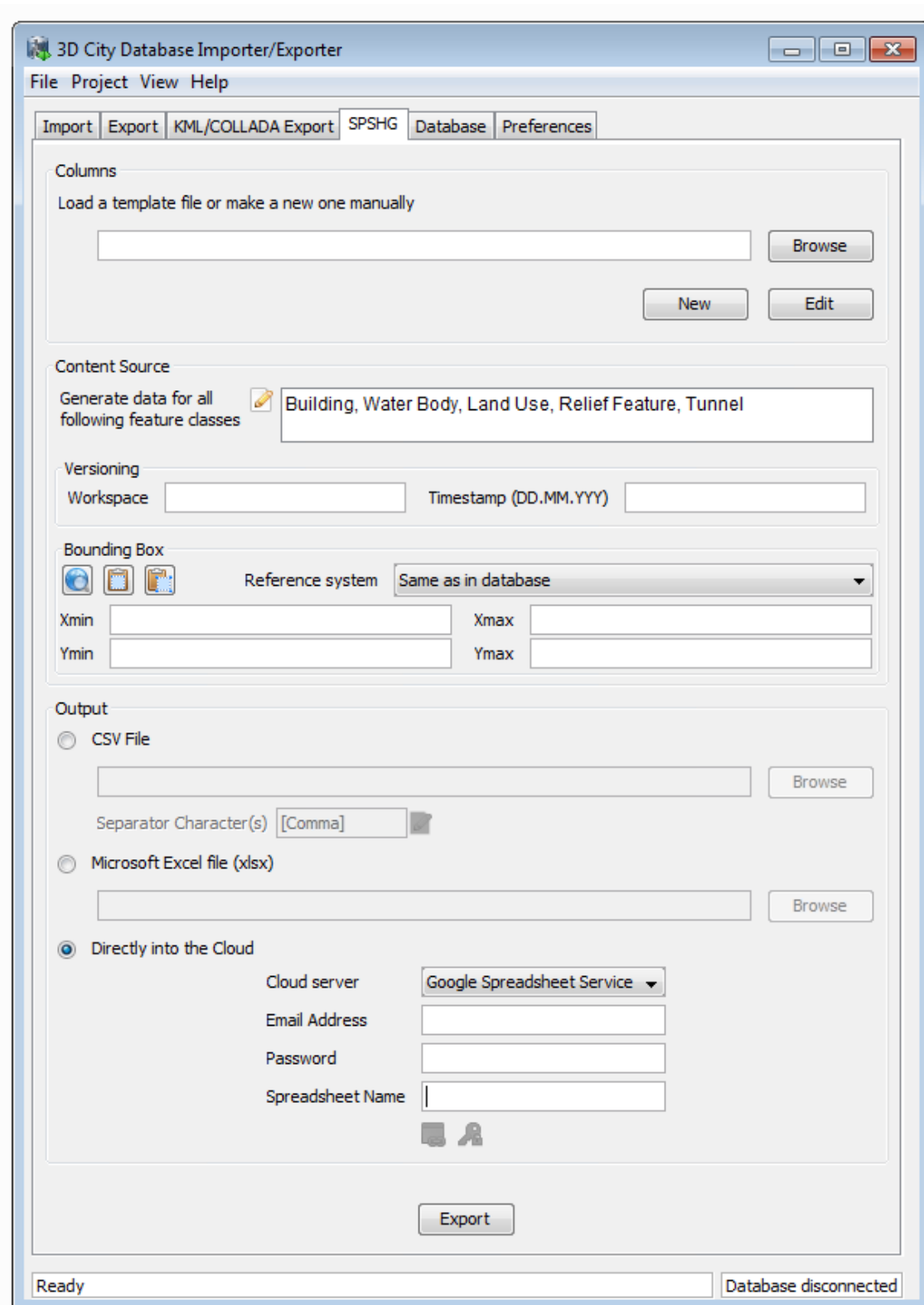


Figure 121: Importer/Exporter started with the *Spreadsheet Generator* plugin installed

6.2 Spreadsheet Generator Plugin (SPSHG)

6.2.1 Definition

By using the SPSHG (Spreadsheet Generator) plugin, it is possible to export data from a 3D City Database (3DCityDB) instance into a CSV file or into an online spreadsheet hosted in a cloud service. The CSV file may be imported to either a spreadsheet application (like Microsoft Excel or Open Office Calc) or to a web based online spreadsheet service (like Google Docs). All features of spreadsheet programs, like calculation and graphing tools, are applicable to the exported data from a 3D City Database instance.

6.2.2 Plugin installation

The SPSHG is an additional component which can be installed together with the 3DCityDB Importer/Exporter tool. During the Installation of the Importer/Exporter tool, the wizard will ask you if you want to install Spreadsheet Generator Plugin like in the following figure:

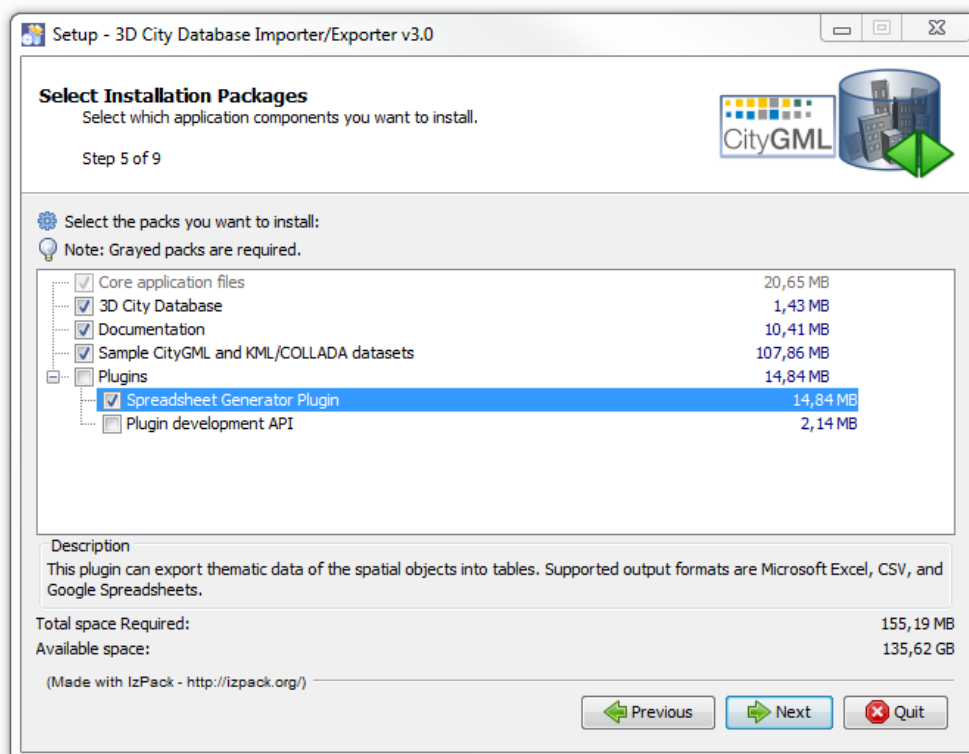


Figure 122: Installation wizard of Import/Export tool (Step 5)

If you haven't checked the "Spreadsheet Generator Plugin" box during the installation process, it is also possible to install the SPSHG later. Following simple steps will guide you through the install process:

- Download the SPSHG plugin zip file from the **official website of the 3D City Database at [www.3dcitydb.org]**.
- Open the folder that contains your locally installed instance of the *Importer/Exporter version 3.0.0* (the installation directory).

- Open the *plugins* subfolder. If it is not available, create a new subfolder and name it “plugins”.
- Extract the downloaded SPSHG plugin zip file in the *plugins* folder. As a result a new folder named *spreadsheet_Generator* will be created. The *spreadsheet_Generator* folder will contain all required files and subfolders.
- Run the *Importer/Exporter*. The SPSHG plugin tab should be visible like in Figure 123.

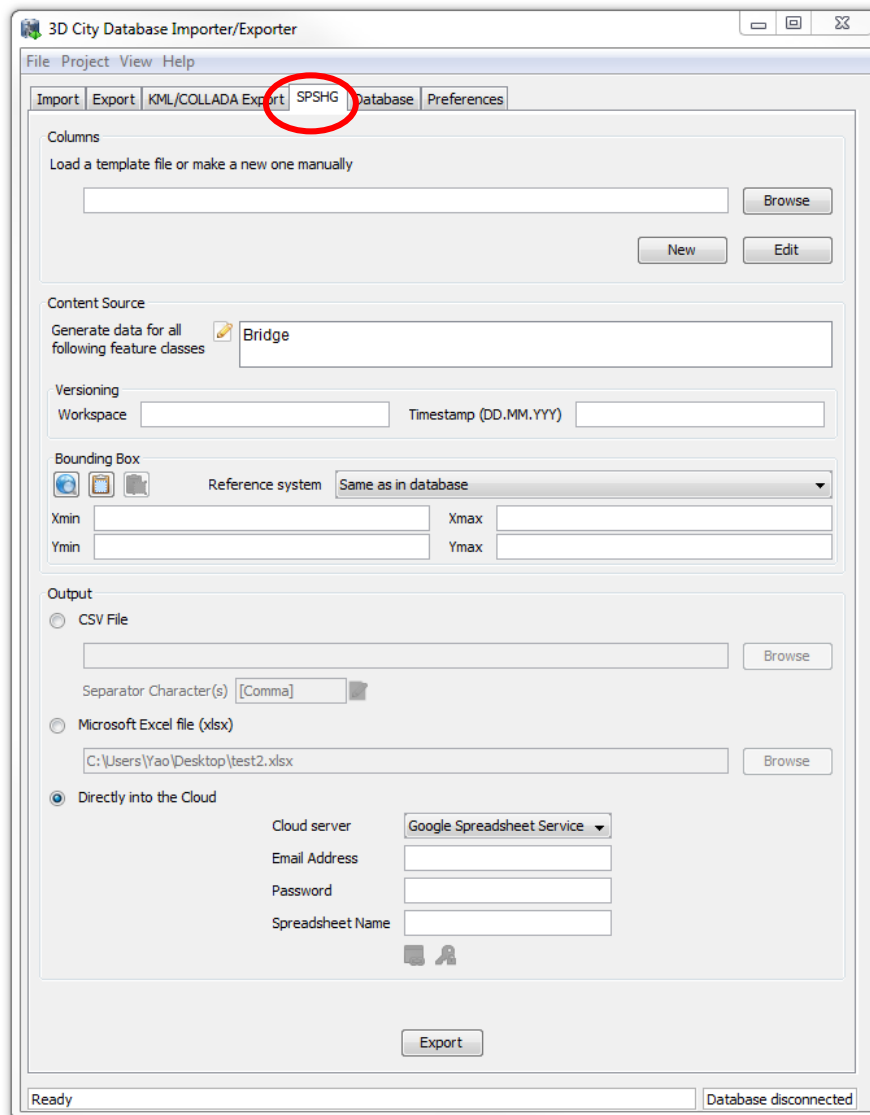


Figure 123: The SPSHG plugin tab allowing for exporting from the 3DCityDB to a spreadsheet.

6.2.3 User Interface

6.2.3.1 Main Parameters

The SPSHG plugin GUI is divided into three main parts. The upper part, titled *Columns*, refers to the columns of the output spreadsheet file. The *Content Source* in the middle section refers to the rows of the output spreadsheet. Each output row will always contain the GMLID of a city

object and its corresponding selected values for each column. A list of the feature classes of city objects (Top-level features) whose data will be exported to the spreadsheet, the versioning information of the database and a geographic bounding box should be specified. Destination of export must be entered in the lower part. It may be sent to a local CSV/XLSX file on the client machine or directly uploaded to a cloud server (Google Drive). All input data fields of the SPSHG plugin tab will be now described in more detail.

6.2.3.2 Columns

First of all, the columns of your resulting spreadsheet should be defined. You can choose to load a template file or manually create a new one:

Load a template file: type the template file's path directly into the text field or click on the *Browse* button to use an *Open* dialog for selecting the template file. The selected template file can be edited by clicking on the *Edit* button.

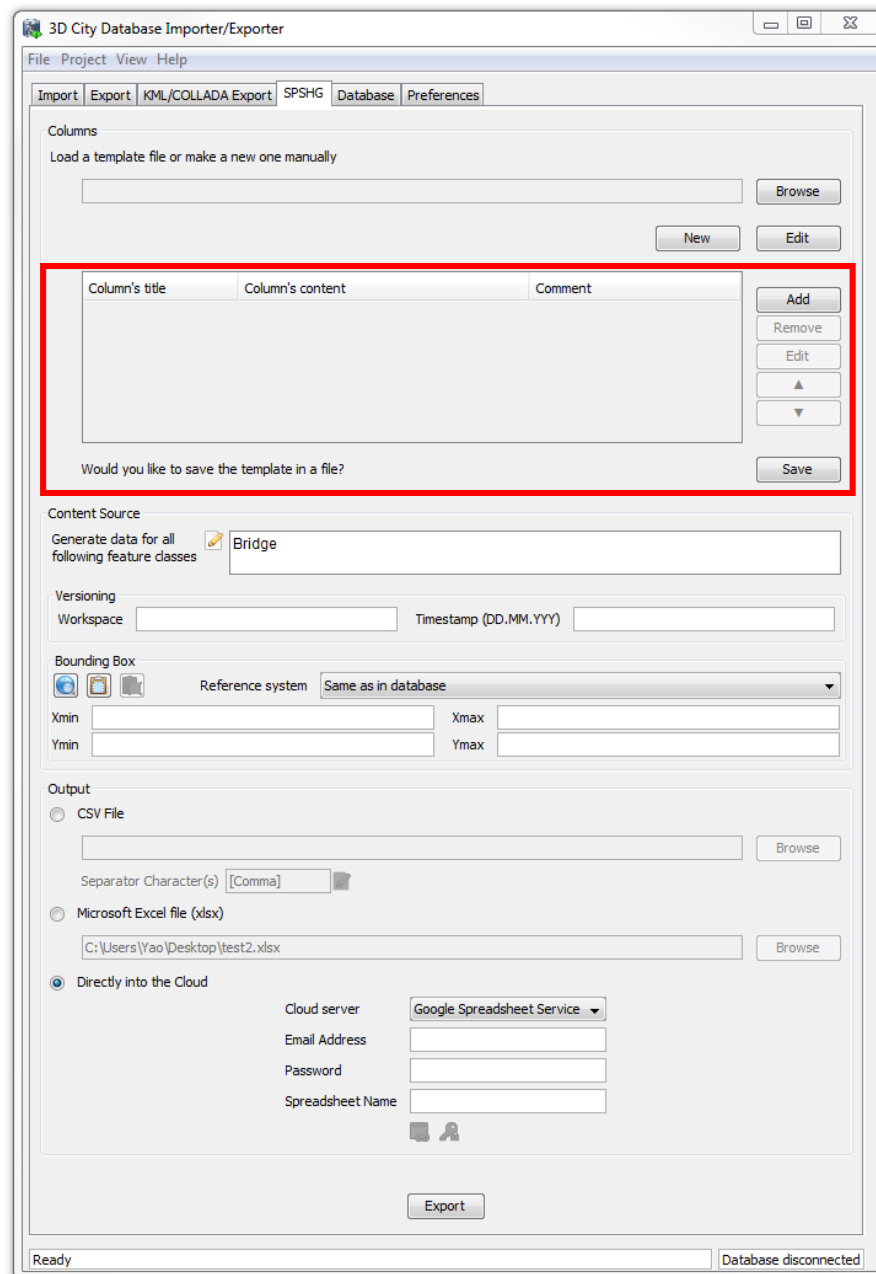


Figure 124: The part for manually creating a new template will appear when clicking on the *New* button. This part will also be shown when clicking on the *Edit* button after a template file is selected.

Create a new template: click on the *New* button to access the part for creating a template (marked in Figure 124). To add a new column click on the *Add* button and fill all necessary fields of the *New Column* dialog. A column contains a *title*, *content* and *comment*. The comment field is optional. Each row in the exported data will begin with the GMLID of the corresponding city object. It will be followed by the adapted value of each column for that city object (see next section for more information). Created columns will be listed in the table. You can use the *Remove*, *Edit*, *Up* (▲), and *Down* (▼) buttons to modify listed columns on the table and their order. By pressing the *Save* button, manually created (or adapted) templates will be saved in a text file. Path will be specified by the *Save* dialog.

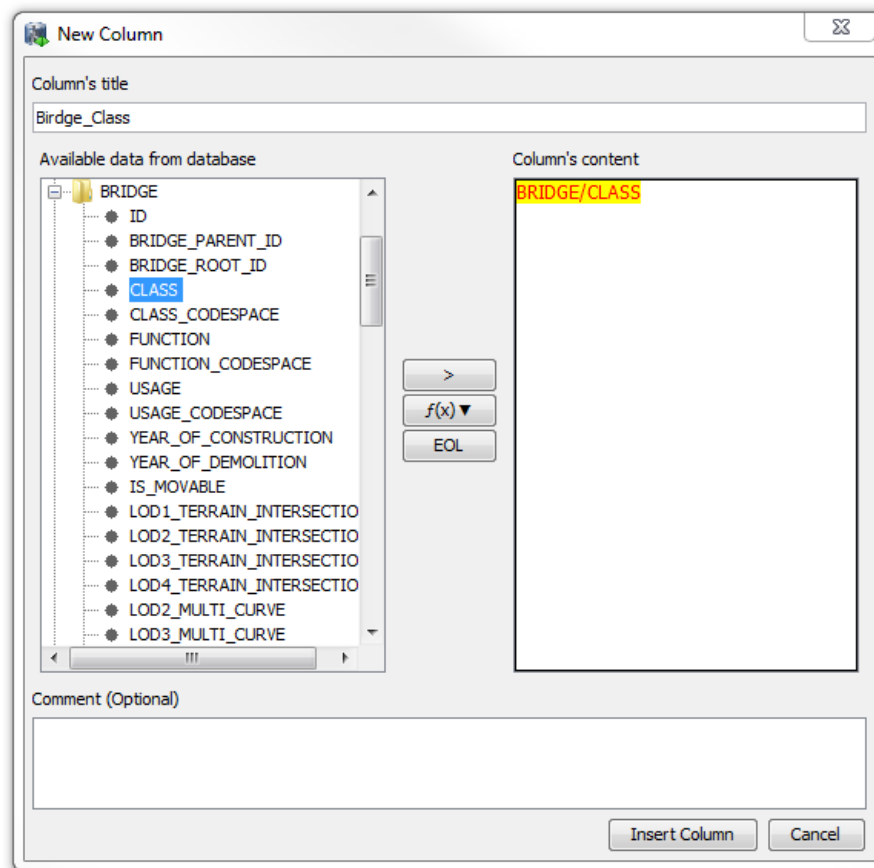


Figure 125: The New Column dialog. Fill the Column's title, Column's content fields and click on the Insert Column button to add it to the list of columns. The Comment field is optional. When written to a template file its content serves informational purposes only.

New Column dialog

By clicking on the *Add* button the *New Column* dialog will be shown (Figure 125). Using the New Column dialog, it is possible to define a new column for the output spreadsheet. A column may contain a *title*, *content* and *comment* fields. The title and content are mandatory. During export time, the content of each column will be adapted for each city object. For each specific column:

- The content may set to be a static value, e.g. *Munich*. As a result, the value of that column in the exported spreadsheet will be equal to the specified static value (in this example *Munich*) for all rows.
- The content of a column may be specified by an expression. The main part of an expression refers to a column in a specific table of a 3D City Database. Each row refers to one city object. Consequently, the value of the spreadsheet's column will be dynamically adapted for each row at export time. It means the value of the spreadsheet's column for a specific row will be equal to the value of that expression for the corresponding city object of that row. Expressions must follow specific rules. They can be added simply by using the GUI or written by hand.

- The content of a spreadsheet's column may contain a combination of static values and expressions.

Rules for Column's Content field

- Expressions are coded in the following form:
"TABLE/[AGGREGATION FUNCTION] COLUMN [CONDITION]".
Aggregation function and condition are optional. Table refers to the underlying 3DCityDB table structure (see Chapter 2.3 for details).
- Expressions are not case-sensitive.
- For each row of output, each expression will only return the value of those entries relevant to the city object for that row. That means an implicit condition clause like "TABLE.CITYOBJECT_ID = CITYOBJECT.ID" is always considered and does not need to be explicitly written.
- In a case that more than one entry for the corresponding city object are available, a comma separated list of values will be returned. When only interested in the first result of a list the aggregation function FIRST should be used. Other possible aggregation functions are LAST, MAX, MIN, AVG, SUM and COUNT.
- Conditions can be defined by a simple number (meaning which element from the result list must be taken) or a column name (that must exist in underlying 3DCityDB table structure) a comparison operator and a value. For instance: [12] or [NAME = 'abc'].
- Invalid results will be silently discarded
- Multiline content is supported. Use "[EOL]" to start a new line in the same column.

How to use the New Column dialog

Title and content of each column should be specified. On the left hand side of the New Column dialog, tables of the 3D City Database and their columns are displayed in a tree structure. Adding an expression is simple. Select a column in a table from the left hand side tree and click on the ">" button. In the case that aggregation functions are needed, select a column from the left hand side tree and click on the $f(x)$ button then chose one of the aggregation functions. As a result of both cases a corresponding expression will be added into the column's content in the right hand side.

A column's content can be several lines long. Write "[EOL]" text in the column's content wherever a new line should be started. You can also press the *EOL* button to automatically add "[EOL]" text to the content. During export time, the "[EOL]" text will be replaced by a new line.

After filling all necessary fields click on the *Insert Column* button. A new column will be created and added to the manually created template.

Examples for Column's Content

ADDRESS/STREET

Returns the content of the STREET column on the ADDRESS table for each city object.
For instance:

Straße des 17. Juni

However ADDRESS table might contain more than one row for some city objects. In such a case a comma separated list of values will be returned. For instance:

Straße des 17. Juni, Straße des 17. Juni, Straße des 17. Juni, Straße des 17. Juni

To avoid that use a proper aggregation function. For instance:

ADDRESS/[FIRST]STREET

Although the ADDRESS table may contain several entries for a city object, result of the above expression will be equal to the street name of first found entry.

**ADDRESS/[FIRST]STREET, ADDRESS/[FIRST]HOUSE_NUMBER
[EOL]ADDRESS/[FIRST]ZIP_CODE ADDRESS/[FIRST]CITY**

Returns the full address of each city object in two lines. For instance:

Straße des 17. Juni, 135
10623 Berlin

CITYOBJECT_GENERICATTRIB/ATTRNAME

Returns the names of all existing generic attributes for each city object. All names will be separated by commas.

CITYOBJECT_GENERICATTRIB/REALVAL[ATTRNAME = 'SOLAR_SUM_INVEST']EUR

Returns the content of the REALVAL column of all existing generic attributes for each city object whose ATTRNAME is equal to 'SOLAR_SUM_INVEST'. The number will be followed by “EUR”. For instance:

23000EUR

Rules for Columns' Template file

Rules for the template file are simple. A template file contains a list of columns and their description. It may be edited by hand or by saving a manually created template.

- A template file is a plain-text file.
- Each row of a template file may describe a column or be a comment.
- Comment rows **MUST** start by // or ;
- A column should be specified in one of following forms:
 - *Title:Content*

Title is the column's title and content is the column's content. In this case title is specified by user.

- *Content*

Content is the column's content. In case the column's title is not specified by the user it will be automatically generated by the plugin.

Example for Template File

Sample template file:

```
// This is a template file for the export of tabular data.  
// Lines starting with // or ; are comments and will be ignored.  
Street:ADDRESS/[FIRST]STREET  
Houseno:ADDRESS/[FIRST]HOUSE_NUMBER  
City:ADDRESS/[FIRST]CITY  
Address:ADDRESS/[FIRST]STREET,  
ADDRESS/[FIRST]HOUSE_NUMBER[EOL]ADDRESS/[FIRST]CITY  
// INVEST  
Investment:CITYOBJECT_GENERICATTRIB/REALVAL[ATTRNAME =  
'SOLAR_SUM_INVEST'] EUR
```

Figure 126 shows a sample export result.

Sample_Template ☆

Name Surname ▾

Share

File Edit View Insert Format Data Tools Help All changes saved

fx | Straße des 17. Juni, 135

Show all formulas X

	A	B	C	D	E	F
1	GMLID	Street	Houseno	City	Address	Investment
2	BLDG_0003000f0028da8a	Straße des 17. Juni	136	Berlin	Straße des 17. Juni, 136 Berlin	315700 EUR
3	BLDG_000300000008f6df	Straße des 17. Juni	115	Berlin	Straße des 17. Juni, 115 Berlin	0 EUR
4	BLDG_0003000f00250727	Straße des 17. Juni	118	Berlin	Straße des 17. Juni, 118 Berlin	263550 EUR
5	BLDG_000300000008f309	Straße des 17. Juni	124	Berlin	Straße des 17. Juni, 124 Berlin	38850 EUR
6	BLDG_0003000e00a0e27c	Straße des 17. Juni	152	Berlin	Straße des 17. Juni, 152 Berlin	444500 EUR
7	BLDG_0003000f0025072f	Straße des 17. Juni	144	Berlin	Straße des 17. Juni, 144 Berlin	493850 EUR
8	BLDG_0003000a001ce4b3	Hardenbergstr.	36	Berlin	Hardenbergstr., 36 Berlin	374150 EUR
9	BLDG_0003000f0093e86	Straße des 17. Juni	145	Berlin	Straße des 17. Juni, 145 Berlin	887950 EUR
10	BLDG_0003000f0093e88	Hardenbergstr.	40A	Berlin	Hardenbergstr., 40A Berlin	107100 EUR
11	BLDG_0003000e00858128	Hardenbergstr.	38	Berlin	Hardenbergstr., 38 Berlin	56700 EUR
12	BLDG_0003000000f51d29	Hardenbergstr.	36	Berlin	Hardenbergstr., 36 Berlin	0 EUR
13	BLDG_0003000e00858120	Hardenbergstr.	36A	Berlin	Hardenbergstr., 36A Berlin	399350 EUR
14	BLDG_0003000000f521db	Straße des 17. Juni	135	Berlin	Straße des 17. Juni, 135 Berlin	0 EUR
15	BLDG_0003000e00206fa4	Hardenbergstr.	128	Berlin	Hardenbergstr., 128 Berlin	124600 EUR
16	BLDG_0003000f0093e80	Ernst-Reuter-Platz	1	Berlin	Ernst-Reuter-Platz, 1 Berlin	233800 EUR
17	BLDG_0003000e00206fbe	Hardenbergstr.	41	Berlin	Hardenbergstr., 41 Berlin	31150 EUR
18					Straße des 17. Juni, 135	

+ Sheet 1 ▾

Straße des 17. Juni, 135

Figure 126: Example of exported data based on sample template presented above from a 3D City Database instance.

6.2.3.3 Content Source

In this GUI section the feature class of city objects and their origin (versioning information and geographic bounding box) should be specified.

Feature Classes

City objects of the selected feature class(es) will be exported. Click on the edit button (marked by 1 in Figure 127) to insert or remove a feature class.

Versioning

Oracle's Workspace Manager enables storing of different versions of the database as named workspaces. The export process will use the specified workspace.

If version management is disabled or the current state of the database should be exported, the default workspace name LIVE must be entered and the timestamp field must remain empty.

Unfortunately, as PostgreSQL does not officially offer any equivalent facility like Workspace Manager, the corresponding elements in the graphical user interface will be disabled whenever the PostgreSQL/PostGIS database instance is connected.

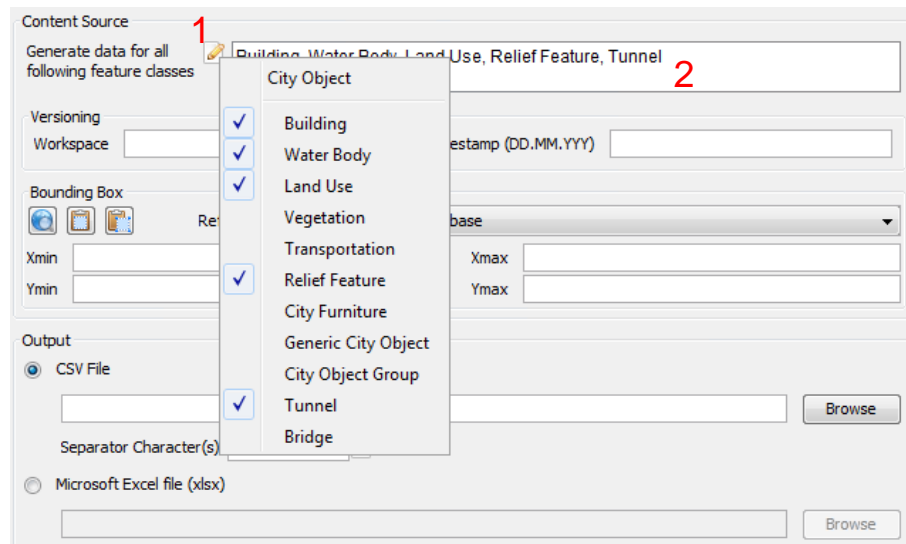


Figure 127: Click on the *edit* button (marked by 1) to add or remove a CityGML feature class from the list of features classes (marked by 2).

Bounding Box

Use the bounding box section to select an area of interest from which the selected features contained should be exported. Insert lower left and upper right coordinates of the bounding box or click on the map button to select the area from a map. Please refer to [Chapter 5.2.2] for more details on the different options for specifying a bounding box.

6.2.3.4 Output

It is possible to export the data in a CSV or XLSX file on the local computer, or directly into an online spreadsheet hosted in a cloud service.

CSV/XLSX File

A CSV/XLSX file is supported by most spreadsheet applications. It can be easily imported into a local spreadsheet processing program like Microsoft Excel and Open Office Calc or to a web based online spreadsheet service like Google Docs.

Click on the *CSV File* or *XLSX file* radio button, and write an output file path or select an output file by clicking on the *Browse* button. It is also possible to specify another separator character(s) instead of comma (default) for CSV file. Write any arbitrary separator phrase or click on the *edit* button (marked by 1 in Figure 128) to select it from a list.

Figure 128: Click on the *CSV File* radio button and write any output file path or click on the *Browse* button to select an output file. Type the separator character (s) or click on the *edit* button (marked by 1) and select one from a list.

Directly into the Cloud

The SPSHG plugin is able to export data directly into an online spreadsheet hosted in a cloud service. Currently **only Google Spreadsheet** is supported. All you need is a Google Docs account. Click on the *Directly into the Cloud* radio button to enable this part. Choose a cloud server and insert the *Email Address* and *Password* of your Google Docs account. Specify a *Spreadsheet Name*. The exported spreadsheet will be shown later by this name in your documents list.

Figure 129: After authentication fails for several times, a CAPTCHA will be shown. Check your email address and password and write what you see in the picture (marked by 1) into the text field (marked by 2). Click on the *Export* button again.

By clicking on the *Export* button the SPSHG plugin will automatically authenticate your *Email Address* and *Password* by contacting to the Google Doc service and a corresponding message will be shown in the console. If the authentication fails several times a CAPTCHA will be shown (Figure 129). To retry, first check your *Email Address* and *Password* are properly written and then write the letters seen in the picture (marked by 1) into the text field (marked by 2) below the picture and try again.

After finishing the export process, the *Link Copy* and *Sharing Settings* buttons will become enabled (marked by 3 and 4 respectively). By clicking on the *Link Copy* button the URL address of the uploaded spreadsheet will be copied into the clipboard. Clicking on the *Sharing Settings* button will open the *Sharing Settings* dialog. See more information in section 6.2.3.6.

6.2.3.5 Export

By clicking on the *Export* button, the export process will begin. If *Directly into the Cloud* is selected as the output form, the inserted *Email Address* and *Password* will be used for authentication and the corresponding message will be shown in the console. Then a connection to a database with the settings from the database tab will be established and export will be started. The output is stored in a temporary file¹ on the user's computer and then uploaded into the cloud. After the export is finished a summary will be shown in console and the temporary file is removed.

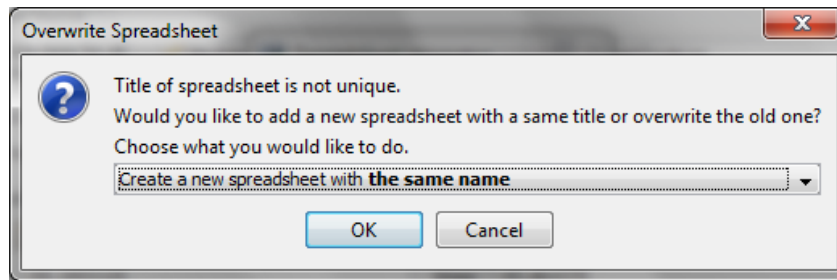
Overwriting

When exporting data directly into the cloud it is possible to overwrite an already existing spreadsheet document if there is already another spreadsheet with the same title stored in the cloud. As a result all settings of the old document will remain unchanged and its content will be updated. It means sharing settings, visibility and document link will not be altered. Whenever there is a spreadsheet name conflict, the program will ask you whether you wish to create a new spreadsheet with that title or overwrite an existing spreadsheet. Before uploading the spreadsheet a dialog as shown in Figure 130 will pop up. You can choose to create a new spreadsheet with the same name, or overwrite one of existing files. When overwriting, there may be more than one spreadsheet with the same name; all spreadsheets carrying this name will be shown sorted by their last modification time and last editing person.

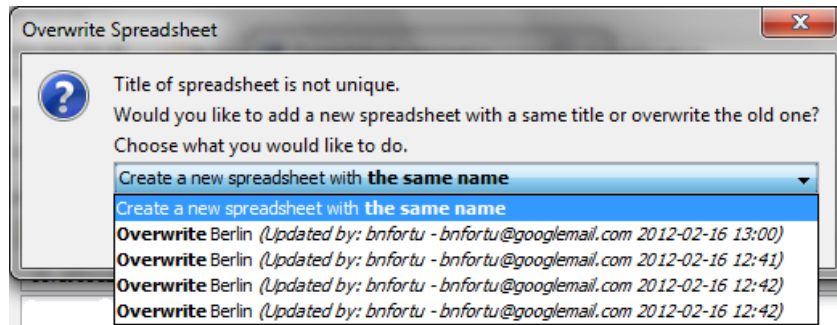
6.2.3.6 Sharing settings

After a successful export into an online spreadsheet hosted in a cloud service, it is possible to change sharing settings of your spreadsheet. Click on the *Sharing Settings* button (marked by 4 in Figure 129) to open the *Sharing Settings* dialog (Figure 131).

¹ The temporary file will be created in the system-dependent default temporary-file directory. On UNIX systems it is typically `"/tmp"` or `"/var/tmp"`; on Microsoft Windows systems it is typically defined by the environment variable `TMP` or `TEMP`.



(a)



(b)

Figure 130: Overwrite spreadsheet dialog. In the case that a document with the same title is existing on your account, this dialog will be shown. It is possible to create a new spreadsheet with the same name, or to choose one of the existing documents to be overwritten. (b) Shows there may be more than a document with the same name (e.g. Berlin). By choosing one of the available spreadsheets its content will be updated.

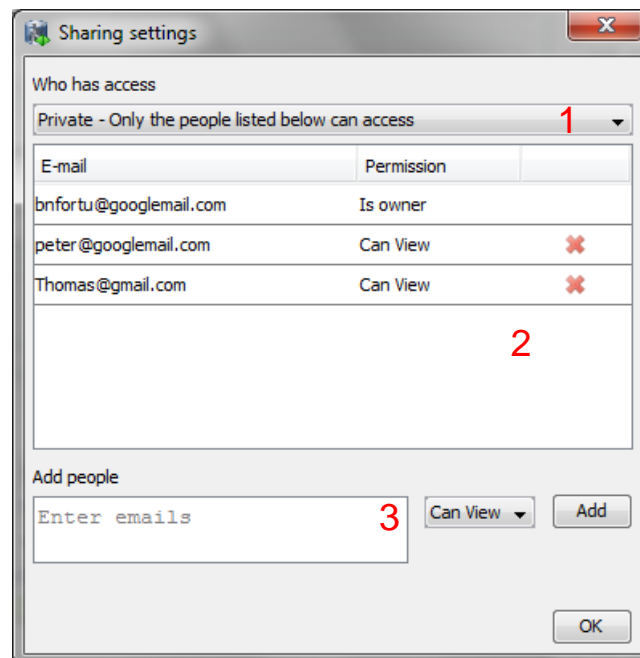


Figure 131: Sharing settings. Use this dialog to globally modify access to the current document including changing visibility settings and giving people access rights

By using the *Sharing settings* dialog it is possible to share a document with many or just a few people by assigning a visibility option to the document. There are three different visibility options: *private*, *anyone with the link*, and *public on the web*. Furthermore, the document can

be shared with selected people as well. Their e-mail address and a sharing rule must be specified.

Visibility Options

Shows who can view or edit this spreadsheet document (Figure 131, marked by 1).

- *Private - Only the people listed below can access*: when you create a private spreadsheet, just people listed in the table, marked by 2, can have access.
- *Anyone who has the link can view/edit*: people can only view the document if they have the exact URL of the spreadsheet. If you select the *Anyone who has the link can edit* option, anyone with the URL will also be able to *edit* your spreadsheet.
- *Public on the web - Anyone on the Internet can find and view/edit*: public spreadsheets may show up in search results, and anyone who finds the web address of the spreadsheet can access it. If you select *Anyone on the Internet can find and edit* option, any person who finds the spreadsheet document will also be able to view, comment and edit it.

Note: People who are listed in the table have already access to the spreadsheet, regardless of the selected visibility option.

Table of selected people

People who are listed in the table (marked by 2 in Figure 131) have access and their own permission regardless of the visibility option. Their names or email addresses are listed in the table followed by their permission and a delete icon (red cross⁹). To modify access of one specific user click on his/her permission in the table or click on the delete icon to remove him/her from the list. As it was explained before; access rights for all people not listed in the table follow the visibility condition of the document.

Add People

Just click on the text area below 'Add people' (marked by 3 in Figure 131) and type any email address, choose their permission and click on Add button. To add more people with just one click write a list of their emails separated by commas.

6.2.4 Example

Here is a step by step guide for creating a spreadsheet, uploading it directly into cloud, and sharing it. Assume the spreadsheet should contain the addresses of all a buildings in a specific area.

- Open the *Importer/Exporter* and select *SPSHG* tab (Figure 123).
- First of all, columns of the spreadsheet should be defined by a template. There are two different ways to do that. Please follow one of them:
 - Manually Creating a Template:

- Click on the *New* button to open the section for creating a template manually (Figure 124).
- Click on the *Add* button to open *New Column* dialog (Figure 125).
- Write the *Column's Title*. In this case type „Address“.

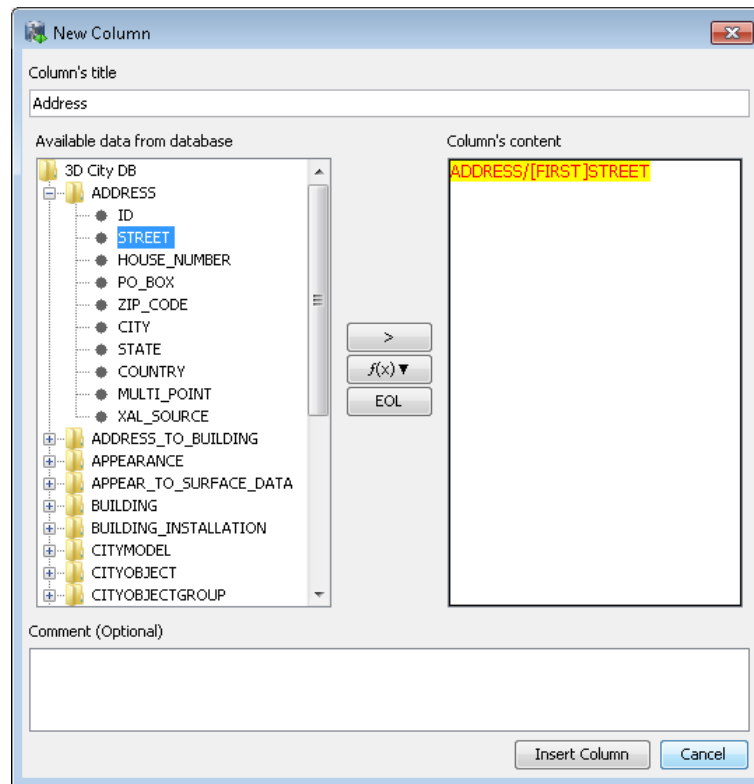


Figure 132: New Column dialog.

- From the left hand *tree* click on *ADDRESS* and then *STREET*. Click on *f(x)* button and choose *FIRST*. As a result the *Column's Content* should be equal to the column's content illustrated in Figure 132.
- Repeat the previous step to add *HOUSE_NUMBER*, and then click on *EOL* button to add new line and repeat it once again to add *CITY*. The *Column's Content* should be equal to the column's content illustrated in Figure 133.

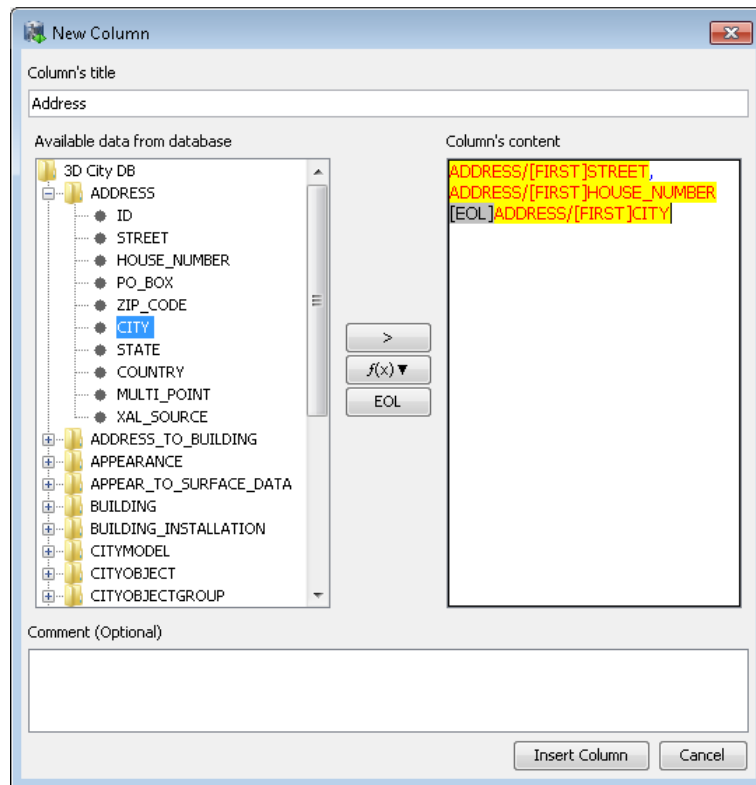


Figure 133: New Column dialog.

- Click on the *Insert Column* button to close the dialog and add current column into the table (Figure 134).
- Loading a template:
 - Create a new plain-text file in your computer and set its name “sample_template.txt”. Copy and paste the content of the following box into that file and save it.

```
ADDRESS/[FIRST]STREET, ADDRESS/[FIRST]HOUSE_NUMBER
[EOL]ADDRESS/[FIRST]CITY
```

- On the *SPSHG* tab of *Importer/Exporter* click on the *Browse* button. Find the “sample_template.txt” file and select it. Then press the *Open* button.
- (Optional) Click on the *Edit* button to see columns of the selected template file. It should be similar to Figure 134.

Columns

Load a template file or make a new one manually

Browse

New Edit

Column's Title	Column's Content	Comment
Address	ADDRESS/[FIRST]STREET, ADDRESS/[FIR...	

Add

Remove

Edit

▲

▼

Would you like to save the template in a file?

Save

Figure 134: List of current columns shown in the table. Double click on a column or press the edit button to modify that column's content. Use the *Save* button to store the current template into a text file.

- Content Source
 - Click on the *edit* button of feature classes list (marked by 1 in Figure 135). Select *Building* on the pop-up menu and unselect the rest. Click on the list (marked by 2 in Figure 135) to close the pop-up menu. The list should just contain *Building* like in Figure 135.
 - Set proper settings for *Workspace* and *Timestamp* in the database *Versioning* part, if required for your database (only supported by Oracle).
 - Use the *Bounding Box* part to select your desired area. Insert lower-left and upper-right corner coordinates or use the *Map Viewer* dialog. Be sure that the database connection settings in database tab point to the relevant database.

Content Source

Generate data for all following feature classes

1 Building 2

Versioning

Workspace LIVE Timestamp (DD.MM.YYY)

Bounding Box

Reference system WGS84

Xmin 13.3187771 Xmax 13.3372307

Ymin 52.5102922 Ymax 52.5175527

Figure 135: Sample settings for *Content Source*

- Output

- Click on the *Directly into the Cloud* radio button.
- Choose a *Cloud Server* like *Google Spreadsheet Service*.
- Insert your *Email address* and *Password* for the selected cloud server. If you chose Google Spreadsheet Service, you just need an ordinary Google account.
- Write a title for the spreadsheet in the *Spreadsheet Name* field, e.g. “sample_upload”. Figure 136 shows a snapshot of proper settings for the output part.

The 'Output' dialog box has three radio buttons: 'CSV File', 'Microsoft Excel file (xlsx)', and 'Directly into the Cloud'. The 'Directly into the Cloud' option is selected. Below it, there are four input fields: 'Cloud server' (a dropdown menu showing 'Google Spreadsheet Service'), 'Email Address' (a text box with 'bnfortu@googlemail.com'), 'Password' (a text box with masked characters), and 'Spreadsheet Name' (a text box with 'sample_upload'). There are also 'Browse' buttons next to the first two input fields. At the bottom, there are two small icons: a folder and a person.

Figure 136: Sample settings for *Output*.

The console window displays the following log messages:

```
[17:48:31 INFO] Authentication is successfully done.
[17:48:31 INFO] Initializing...
[17:48:33 INFO] Exported CityGML features:
[17:48:33 INFO]      Building: 231
[17:48:33 INFO] All in all 231 CityObjects are exported.
[17:48:37 INFO] The data file is successfully uploaded.
Address:https://docs.google.com/spreadsheet/ccc?key=0Atucs0E183jCdERaY2FjdHROR
FlfY1F4WFg4eXhwZ0E
[17:48:38 INFO] Spreadsheet export successfully finished.
```

Figure 137: Console logs for a successful export and upload.

- Export
 - Click on the *Export* button to start the export process.
 - Look for relevant messages in the *console*. Figure 137 shows a snapshot of a successful export.
- (Optional) Change Sharing Settings:
 - Click on the *Sharing Settings* icon (marked by 4 in Figure 129) to open the *Sharing Settings* dialog (Figure 138).
 - Change the *Visibility Option* (marked by 1 in Figure 138) from *Private - Only the people listed below can access* to *Anyone who has the link can view*.
 - Press the OK button.
 - Click on the *Link Copy* icon (marked by 3 in Figure 129) to copy the link of your uploaded spreadsheet into the clipboard.
 - Use a web-browser to see the link.

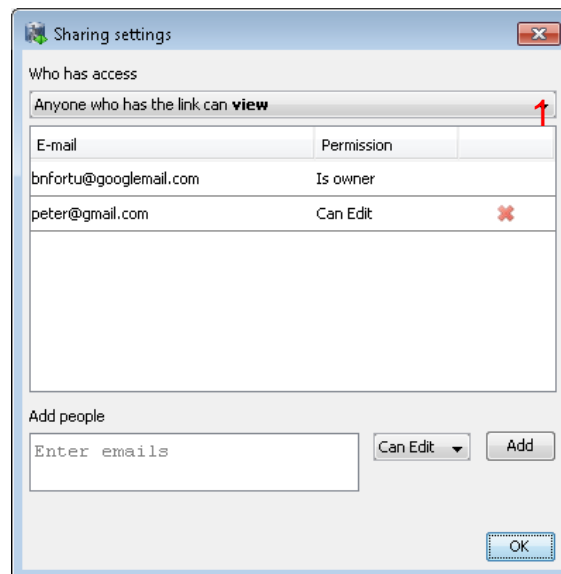


Figure 138: Sharing settings dialog. Visibility is set to *Anyone who has the link can view*. In addition to the owner another user can also edit the spreadsheet.

7 Web Feature Service

The OGC *Web Feature Service* Interface Standard (WFS) provides a standardized and open interface for requesting geographic features across the web using platform-independent calls. Rather than sharing geographic information at the file level, for example, the WFS offers direct fine-grained access to geographic information at the feature and feature property level. Web feature services allow clients to only retrieve or modify the data they are seeking, rather than retrieving a file that contains the data they are seeking and possibly much more.

The 3D City Database offers a Web Feature Service interface allowing web-based access to the 3D city objects stored in the database. WFS clients can directly connect to this interface and retrieve 3D content for a wide variety of purposes. Thus, users of the 3D City Database are no longer limited to using the Importer/Exporter tool for data retrieval. The WFS interface is platform-independent and database-independent, and therefore can be easily used to build CityGML-aware applications.

The 3D City Database WFS interface is implemented against the latest *version 2.0* of the OGC Web Feature Service standard (OGC Doc. No. 09-025r2) and hence is compliant with ISO 19142:2010. Previous versions of the WFS standard are not supported though. The implementation currently satisfies the *Simple WFS* conformance class. The development of the WFS is led by the company *virtualcitySYSTEMS GmbH*, Berlin, which offers an extended version of the WFS with additional functionalities that go beyond the *Simple WFS* class (e.g., thematic and spatial filter capabilities and transaction support). This additional functionality may be fed back to the open source project in future releases.



The 3D City Database Web Feature Service is free software under the *GNU Lesser General Public License version 3.0*. See the `LICENSE` file shipped with the software for more details. For a copy of the GNU Lesser General Public, please visit <http://www.gnu.org/licenses/>.

7.1 System requirements

The 3D City Database WFS is implemented as *Java web application* based on the *Java Servlet* technology. It therefore must be run in a Java servlet container on a web server. The following minimum *software requirements* have to be met:

- **Java servlet container** supporting the **Java Servlet 3.0** specification
- **Java 7 or 8** Runtime Environment (Java 6 or earlier versions are not supported)

The WFS implementation has been successfully deployed and tested on **Apache Tomcat 7** (<http://tomcat.apache.org/>). This is also the *recommended* servlet container. *Apache Tomcat 8* is also supported, whereas any previous version of the Apache Tomcat server will not work.

Note: Neither Java nor a servlet container are part of the WFS distribution package and therefore must be properly installed and configured before deploying the WFS. Please refer to the documentation of your favorite servlet container for more information.

Hardware requirements for the web server running the WFS depend on the intended use and number of concurrent accesses. There are no minimum requirements to be met, so make sure your system setup meets your needs. Also note that the WFS **does not provide its own security layer** (e.g., to limit access to specific networks or users). So it is your responsibility to take any reasonable physical, technical and administrative measures to secure the WFS service and the access to the 3D City Database.

WFS clients connecting to the WFS interface of the 3D City Database must support the *OGC WFS standard version 2.0*. Moreover, they must be capable of consuming 3D data encoded in CityGML, which is delivered by the WFS server.

7.2 Installation

The 3D City Database WFS is shipped as a Java WAR (web archive) file. Please download the WFS distribution package from <http://www.3dcitydb.org/>. Besides the WAR file, the distribution package also contains Java libraries that render mandatory dependencies for the WFS service and that must be installed as shared libraries in your servlet container.

Note: Alternatively, you may build your own WAR file from the source code provided on GitHub (<https://github.com/3dcitydb/web-feature-service>). This requires that you are experienced in building Java web applications from source. No further documentation is provided here.

Please follow the following installation steps:

Step 1: Install and properly configure your Java servlet container.

Please refer to the documentation of your servlet container for hints on installation and configuration. Make sure that the servlet container uses Java 7 (or higher) for running web applications.

Step 2: Install the mandatory JAR libraries in your servlet container.

The WFS service requires mandatory JAR libraries to be available in the servlet container. This mainly comprises JDBC libraries for connecting to the database system running the 3D City Database instance. The libraries are shipped with the distribution package. The list of libraries will look similar to this:

- `ojdbc7.jar` (Oracle JDBC driver)
- `postgresql-9.4-1200.jdbc41.jar` (PostgreSQL JDBC driver)
- `postgis-jdbc-2.1.5.jar` (PostGIS JDBC extension)

The libraries have to be installed as *shared libs* or *common libs* (terminology may differ) in your servlet container. For Apache Tomcat 7 (or higher), this simply means placing the JAR files into the `lib` folder of the Tomcat installation directory. Afterwards, you need to restart Tomcat. Please refer to the documentation of your servlet container for more information.

Step 3: Deploy the WFS WAR file on your servlet container.

If your servlet container is correctly set up and configured, simply deploy the WAR file in order

to install the WFS web service. Again, the way how to deploy a WAR file varies for different servlet containers. For Apache Tomcat servers, copy the WAR file into the `webapps` folder, which, per default, is located in the installation directory of the Apache Tomcat server. This will automatically deploy the application. Alternatively, use the web-based Tomcat *manager application* to deploy WAR files on the server. The manager application is included in a default installation. For more information on deploying WAR files on Tomcat or different servlet containers, please refer to the corresponding documentation material.

Note: If you use the automatic deployment feature of Tomcat as described above, the name of the WAR file will be used as *context path* in the URL for accessing the application. For example, if the WFS WAR file is named `citydb-wfs.war`, then the context path of the WFS service will be `http://[host][:port]/citydb-wfs/`. To pick a different context path, simply rename the WAR file or change Tomcat's default behavior.

Step 4: Configure your servlet container.

Make sure that your servlet container has enough memory assigned (heap space ~ 1GB or more, perm space ~ 512MB).

Step 5: Configure the WFS service.

As a last step, the WFS has to be configured to meet your needs. For instance, this includes providing connection details for the 3D City Database instance and the definition of the feature types that shall be served through the interface. These settings have to be manually edited in the configuration file `config.xml` of the service. A graphical user dialog will be developed for a future release. Please check the next chapter for how to configure the WFS.

Note: Changes to the `config.xml` file typically *require a reload or restart* of the WFS web application (a restart of the servlet container itself is, of course, not required). Please check to documentation of your favorite servlet container for how to do so. In case of Apache Tomcat, you can simply use the *manager application* to reload web applications.

7.3 Configuring the Web Feature Service

After deploying but before using the WFS service, you need to edit the `config.xml` file to make the service run properly. The `config.xml` file is located in the `WEB-INF` directory of the WFS web application. If you use Apache Tomcat, `WEB-INF` is a subfolder of the *application* folder, which is generally named after the WAR file and itself is a subfolder of the `webapps` folder in the Tomcat installation directory. This may be different if you use another servlet container.

For example, assume that the WFS web application was deployed under the context name `citydb-wfs`. Then the location of the `WEB-INF` folder and the `config.xml` file in a default Apache Tomcat installation is shown below.

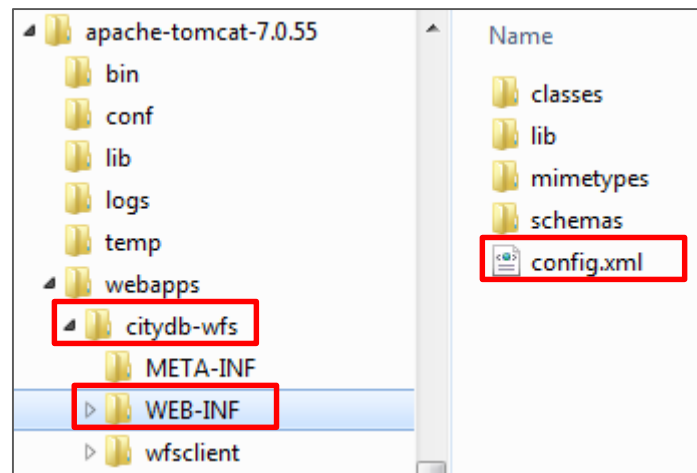


Figure 139: Location of the WEB-INF folder and the config.xml file.

Open the `config.xml` file with a text or XML editor of your choice and manually edit the settings. An XML Schema for validating the contents of the `config.xml` file is provided as file `config.xsd` in the subfolder `schemas`. **After every edit** to the `config.xml` file, **make sure** that the `config.xml` file **validates against this schema** before reloading the WFS web application. Otherwise, the application might refuse to load or unexpected behavior may occur.

In the `config.xml` file, the WFS settings are organized into the main XML elements `<capabilities>`, `<featureTypes>`, `<operations>`, `<database>`, `<server>`, `<uidCache>`, `<security>`, and `<logging>`. The discussion of the settings follows this organization in the subsequent clauses.

7.3.1 Database settings

The *database* settings define the *connection parameters* for connecting to the 3D City Database instance the WFS service should give access to. The contents of the `<database>` element are shown below.

```
<database>
  <connection
    initialSize="10"
    maxActive="100"
    maxIdle="50"
    minIdle="0"
    suspectTimeout="60"
    timeBetweenEvictionRunsMillis="30000"
    minEvictableIdleTimeMillis="60000">
  <description/>
  <type>PostGIS</type>
  <server/>
  <port>5432</port>
  <sid/>
  <user/>
  <password/>
```

```
</connection>
</database>
```

Listing 1: Database settings in the WFS `config.xml` file.

Provide the *type* of the database (Oracle or PostGIS), the *server* name (network name or IP address) and *port* number (default: 1521 for Oracle; 5432 for PostgreSQL) of the database server, the *sid* (when using Oracle, enter the database SID or service name; for PostgreSQL enter the database name), and the *user* and *password* of the database user. You can copy&paste these settings from the config file of the Importer/Exporter. The *description* is optional and can be left empty.

In addition to these minimum settings, the `<connection>` element takes *optional attributes* that let you configure the use of physical connections to the database server. This is especially important for production servers and if more than one WFS service connects to the same database server (in this case, you should also carefully configure the database itself). The attributes together with their meaning are described in the following table.

Attribute	Description
<code>initialSize</code>	(int) the initial number of physical connections that are created when the database connection is established (default: 10).
<code>maxActive</code>	(int) The maximum number of active connections to the database that can be allocated at the same time (default: 100). NOTE – make sure your database is configured to handle this number of parallel active connections.
<code>maxIdle</code>	(int) The maximum number of connections that should be kept active at all times (default: 50). Idle connections are checked periodically (if enabled) and connections that have been idle for longer than <code>minEvictableIdleTimeMillis</code> will be released. (also see <code>testWhileIdle</code>)
<code>minIdle</code>	(int) The minimum number of established connections that should be kept active at all times (default: 0). The connection pool can shrink below this number if validation queries fail. (also see <code>testWhileIdle</code>)
<code>maxWait</code>	(int) The maximum number of milliseconds that the service will wait (when there are no available connections) for a connection before throwing an exception (default: 30000, i.e. 30 seconds).
<code>testOnBorrow</code>	(boolean) The indication of whether connections will be validated before being used by the service. If the connections fails to validate, it will be dropped, and the service will attempt to borrow another (default: false). NOTE - for a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string. In order to have a more efficient validation, see <code>validationInterval</code> .
<code>testOnReturn</code>	(boolean) The indication of whether connections will be validated before being returned to the internal connection pool (default: false). NOTE - for a true value to have any effect, the <code>validationQuery</code> parameter must be set to a non-null string.
<code>testWhileIdle</code>	(boolean) The indication of whether connections will be validated by the idle connections evictor (if any). If a connections fails to validate, it will be dropped (default: false). NOTE - for a true value to have any effect, the

	validationQuery parameter must be set to a non-null string.
validationQuery	(String) The SQL query that will be used to validate connections. If specified, this query does not have to return any data (default: null). Example values are "select 1 from dual" (Oracle) or "select 1" (PostgreSQL).
validationClassName	(String) The name of a class which implements the <code>org.apache.tomcat.jdbc.pool.Validator</code> interface and provides a no-arg constructor (may be implicit). If specified, the class will be used instead of any validation query to validate connections (default: null). NOTE – for a non-null value to have any effect, the class has to be implemented by you as part of the source code of the WFS service. Use with care.
timeBetweenEvictionRunsMillis	(int) The number of milliseconds to sleep between runs of the idle connection validation/cleaner. This value should not be set under 1 second. It dictates how often we check for idle, abandoned connections, and how often we validate idle connections (default: 30000, i.e. 30 seconds).
minEvictableIdleTimeMillis	(int) The minimum amount of time a connection may be idle before it is eligible for eviction (default: 60000, i.e. 60 seconds).
removeAbandoned	(boolean) Flag to remove abandoned connections if they exceed the <code>removeAbandonedTimeout</code> . If set to true a connection is considered abandoned and eligible for removal if it has been in use longer than the <code>removeAbandonedTimeout</code> . See also <code>logAbandoned</code> (default: false).
removeAbandonedTimeout	(int) Timeout in seconds before an abandoned (in use) connection can be removed (default: 60, i.e. 60 seconds). The value should be set to the longest running query.
logAbandoned	(boolean) Flag to log stack traces for application code which abandoned a connection. NOTE - this adds overhead for every connection borrow (default: false).
connectionProperties	(String) The connection properties that will be sent to the JDBC driver when establishing new connections. Format of the string must be <code>[propertyName=property;]*</code> NOTE - The "user" and "password" properties will be passed explicitly, so they do not need to be included here (default: null).
initSQL	(String) A custom query to be run when a connection is first created (default: null).
validationInterval	(long) To avoid excess validation, only run validation at most at this frequency - time in milliseconds. If a connection is due for validation, but has been validated previously within this interval, it will not be validated again (default: 30000, i.e. 30 seconds).
jmxEnabled	(boolean) Register the internal connection pool with JMX or not (default: true).
fairQueue	(boolean) Set to true if connection requests should be treated fairly in a true FIFO fashion (default: true)
abandonWhenPercentageFull	(int) Connections that have been abandoned (timed out) will not get closed and reported up unless the number of connections in use are above the percentage defined by <code>abandonWhenPercentageFull</code> . The value should be between 0-100 (default: 0, which implies that connections are eligible for closure as soon as <code>removeAbandonedTimeout</code> has been reached).
maxAge	(long) Time in milliseconds to keep connections alive. When a connection is returned to the internal pool, it will be checked

	whether now - time-when-connected > maxAge has been reached, and if so, the connection is closed (default: 0, which implies that connections will be left open and no age check will be done).
suspectTimeout	(int) Timeout value in seconds (default: 0).

Table 26: Optional database connection settings.

7.3.2 Capabilities settings

The *capabilities* settings define the contents of the *capabilities* document that is returned by the WFS service upon a `GetCapabilities` request. In general, the WFS implementation supports two ways for generating the capabilities document:

1. **Dynamic generation.** The *capabilities* document is generated dynamically from the contents of the `config.xml` file at request time. Only optional *service metadata* has to be explicitly specified by the user in this case, whereas all other sections of the *capabilities* document are populated automatically from the `config.xml` file. For example, the set of feature types advertised in the `<wfs:FeatureTypeList>` section is derived from the content of the `<featureTypes>` element (cf. chapter 7.3.3). Dynamic generation is the default and *recommended* way.
2. **Static generation.** The *capabilities* document is stored in a local file, which is returned “as is” by the WFS service. Note that settings in the `config.xml` file may easily deviate from the values in a static *capabilities* document. When choosing this option, you have to make sure that the static *capabilities* document conforms to the WFS XML Schema definition.² No check is carried out by the WFS implementation.

If the *capabilities* document shall be generated in a static way, then the `config.xml` file **must contain** a `<capabilities>` node with a `<staticDocument>` child element whose value **shall be an absolute path** pointing to the local XML file storing the *capabilities* document.

```
<capabilities>
  <staticDocument>path/to/the/capabilities.xml</staticDocument>
</capabilities>
```

Listing 2: Specifying a static capabilities document in the `config.xml` file.

Note: A web application typically has limited access to the file system for security reasons. Thus, make sure that the XML file is accessible for the WFS web application. Check the documentation of your servlet container for details.

Instead of the `<staticDocument>` child node, the `<capabilities>` element may also contain a `<owsMetadata>` element as child that allows for specifying *service metadata* (see the example listing below). The metadata is copied to the *capabilities* document “as is”. Thus, the WFS implementation neither performs a consistency check nor validates the provided metadata.

² <http://schemas.opengis.net/wfs/2.0/wfs.xsd>

```

<capabilities>
  <owsMetadata>
    <ows:ServiceIdentification>
      <ows:Title>3DCityDB Web Feature Service</ows:Title>
      <ows:ServiceType>WFS</ows:ServiceType>
      <ows:ServiceTypeVersion>2.0.0</ows:ServiceTypeVersion>
      <ows:ServiceTypeVersion>2.0.2</ows:ServiceTypeVersion>
    </ows:ServiceIdentification>
    <ows:ServiceProvider>
      <ows:ProviderName/>
      <ows:ServiceContact/>
    </ows:ServiceProvider>
  </owsMetadata>
</capabilities>

```

Listing 3: Service metadata settings in the WFS `config.xml` file.

Service metadata comprises, for example, information about the *service itself* that might be useful in machine-to-machine communication or for display to a human. Such information is announced through the `<ows:ServiceIdentification>` child element. In contrast, the child element `<ows:ServiceProvider>` contains information about the *service provider* such as contact information. Please refer to the *OGC Web Services Common Specification* (OGC 06-121r3:2009) to get an overview of the supported metadata fields that may be included in the *capabilities* document and therefore can be specified in `<owsMetadata>`.

Note: Service metadata is *optional* and therefore does not have to be included in the `config.xml` file. Simply provide no content for the `<capabilities>` element or omit it completely. In both cases, the *capabilities* document will nevertheless be generated dynamically.

7.3.3 Feature type settings

With the *feature type* settings, you can control which feature types can be queried from the 3D City Database and are served through the WFS interface. Every feature type that shall be advertised to a client must be explicitly listed in the `config.xml` file.

An example of the corresponding `<featureTypes>` XML element is shown below. In this example, both *building* and *road* objects are available from the WFS service.

```

<featureTypes>
  <featureType>
    <name>Building</name>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180 -90</ows:LowerCorner>
      <ows:UpperCorner>180 90</ows:UpperCorner>
    </ows:WGS84BoundingBox>
  </featureType>
  <featureType>
    <name>Road</name>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180 -90</ows:LowerCorner>
      <ows:UpperCorner>180 90</ows:UpperCorner>
    </ows:WGS84BoundingBox>
  </featureType>
</featureTypes>

```



```

    </ows:WGS84BoundingBox>
  </featureType>
  <version isDefault="true">2.0</version>
  <version>1.0</version>
</featureTypes>

```

Listing 4: Advertised feature types in the WFS `config.xml` file.

The `<featureTypes>` element contains one `<featureType>` node per feature type to be advertised. The feature type is specified through the mandatory *name* property, which can only take values from a fixed list that enumerates the names of the CityGML top-level features (cf. `config.xsd` schema file). In addition, the geographic region covered by all instances of this feature type in the 3D City Database can optionally be announced as *bounding box* (lower left and upper right corner). The coordinate values have to be given in WGS 84.

Note: The bounding box is not automatically checked against or computed from the database, but rather copied to the WFS *capabilities* document “as is”.

Besides the list of advertised feature types, also the CityGML *version* to be used for encoding features in a response to a client’s request has to be specified. Use the `<version>` element for this purpose, which takes either 2.0 (for CityGML 2.0) or 1.0 (for CityGML 1.0) as value. If both versions shall be supported, simply use two `<version>` elements. However, in this case, you should define the *default version* to be used by the WFS by setting the `isDefault` attribute to true on one of the elements (otherwise, CityGML 2.0 will be the default).

7.3.4 Operations settings

The *operations* settings are used to define the operation-specific behavior of the WFS.

```

<operations>
  <useXMLValidation>true</useXMLValidation>
  <GetFeature>
    <outputFormat>application/gml+xml; version=3.1</outputFormat>
    <outputFormat>GML3.1+GZIP</outputFormat>
  </GetFeature>
</operations>

```

Listing 5: Operations settings in the WFS `config.xml` file.

When setting the `<useXMLValidation>` child element to true, all XML encoded operation requests sent to the WFS are first validated against the WFS and CityGML XML schemas. Requests that violate the schemas are *not processed* but instead a corresponding error message is sent back to the client. Although XML validation might take some milliseconds, it is **highly recommended** to always set this option to true to avoid unexpected failures due to XML issues.

With this version of the WFS interface, the only operation that can be further configured is the `<GetFeature>` operation. You can actually choose the available *output formats* that can be used in encoding the response to the client. The value “application/gml+xml; version=3.1” is the default and basically means that the response to a *GetFeature* operation will be purely XML-encoded (using CityGML as encoding format with the version specified in the *feature type* settings, cf. chapter 7.3.3). In addition, the WFS can advertise the output format

“GML3.1+GZIP”. In this case, the response is additionally compressed using GZIP. Since CityGML data quickly grows in size, an additional GZIP compression helps in reducing the amount of data to be sent over the network. So it is **strongly recommended** that the WFS offers this additional output format.

Note: The WFS can only advertise the different output formats in the *capabilities* document. It is up to the client though to actually choose one of these output formats when requesting feature data from the WFS.

7.3.5 Server settings

Server-specific settings are available through the `<server>` element in the `config.xml` file.

```
<server>
  <externalServiceURL>http://yourserver.org/citydb-wfs</externalServiceURL>
  <maxParallelRequests>30</maxParallelRequests>
  <waitTimeout>60</waitTimeout>
</server>
```

Listing 6: Server settings in the WFS `config.xml` file.

The external service URL of the WFS can be denoted using the `<externalServiceURL>` element. The URL should include the *protocol* (typically `http` or `https`), the *server name* and the full *context path* where the service is available for clients. Also announce the *port* on which the service listens if it is not equal to the default port associated with the given protocol.

Note: The service URL is **not configured** through `<externalServiceURL>`. It rather follows from your servlet container settings and network access settings (e.g., if your servlet container is behind a reverse proxy). The `<externalServiceURL>` value is *only used in the capabilities* document and thus announced to a client. Most clients rely on the service URL in the *capabilities* document and will send requests to this URL. So make sure that the WFS is available at the `<externalServiceURL>` provided in the `config.xml`.

The `<maxParallelRequests>` value defines how many requests will be handled by the WFS service at the same time (default: 30). If the number of parallel requests exceeds the given limit, then new requests are blocked until active requests have been fully processed and the total number of active requests has fallen below the limit.

Note: Every WFS can only open a maximum number of physical connections to the database system running the 3D City Database instance. This upper limit is set through the `maxActive` attribute on the `<connection>` element (cf. chapter 7.3.1). Since every request may use more than one connection, make sure that the total number of parallel requests is below the maximum number of physical connections.

In case an incoming request is blocked because the maximum number of parallel requests has been reached, the `<waitTimeout>` option lets you specify the maximum time in seconds the

WFS service waits for a free request slot before sending an error message to the client (default: 60 seconds).

7.3.6 Cache settings

When exporting data, the WFS has to keep track of various temporary information. For instance, when resolving XLinks, the `gml:id` values as well as additional information about the related features and geometries must be available. This information is kept in main memory for performance. However, when memory limits are reached, the cache is written to *temporary tables* in the database.

Per default, temporary tables are created in the *3D City Database instance* itself. The tables are populated during the export operation and are automatically dropped after the operation has finished. Alternatively, the *cache* settings available through the `<uidCache>` element let a user choose to store the temporary information in the *local file system* instead.

```
<uidCache>
  <mode>local</mode>
</uidCache>
```

Listing 7: Cache settings in the WFS `config.xml` file.

The `<mode>` property allows for switching between *database* cache (default) and *local* cache.

Some reasons for using a local, file-based storage are:

- The 3D City Database instance is kept clean from any additional (temporary) table.
- If the Importer/Exporter runs on a different machine than the 3D City Database instance, sending temporary information over the network might be slow. In such cases, using a local storage might help to increase performance.

7.3.7 Security settings

The `config.xml` provides some simple security-related settings with the `<security>` element.

```
<security>
  <maxFeatureCount>10</maxFeatureCount>
  <stripGeometry>>false</stripGeometry>
</security>
```

Listing 8: Security settings in the WFS `config.xml` file.

The `<maxFeatureCount>` property restricts the number of city objects to be returned by the WFS to the user-defined value, even if the request is satisfied by more city objects in the 3D City Database. The default behavior is to return all city objects matching a request. If a maximum count limit is defined, then this limit is automatically advertised in the server's *capabilities* document using the *CountDefault* constraint (given that the capabilities document is created dynamically).

When setting `<stripGeometry>` to *true* (default: *false*), the WFS will remove all spatial properties from a city object before returning the city object to the client. So the client will not receive any geometry values.

Note: The security settings in `config.xml` do not replace a real security layer on user, database or network level. So it is your responsibility to take any reasonable physical, technical and administrative measures to secure the WFS service and the access to the 3D City Database.

7.3.8 Logging settings

The WFS service logs messages and errors that occur during operations to a dedicated log file. Entries in the log file are associated with a timestamp, the severity of the event and the IP address of the client (if available). Per default, the log is stored in the file `WEB-INF/wfs.log` within the *application folder* of the WFS web application.

The `<logging>` element in the `config.xml` file is used to adapt these default settings. The attribute *logLevel* on the `<file>` child element lets you change the severity level for log messages to *debug*, *info*, *warn*, or *error* (default: *info*). Additionally, you can provide an alternative absolute path and filename where to store the log messages.

Note: A web application typically has limited access to the file system for security reasons. Thus, make sure that the log file is accessible for the WFS web application. Check the documentation of your servlet container for details.

If you want log messages to be additionally printed to the console, then simply include the `<console>` child element as well. The `<console>` element also provides a *logLevel* attribute to define the severity level.

```
<logging>
  <console logLevel="info"/>
  <file logLevel="info">
    <fileName>path/to/your/wfs.log</fileName>
  </file>
</logging>
```

Listing 9: Logging settings in the WFS `config.xml` file.

Note: Log messages are continuously written to the same log file. The WFS application does not include any mechanism to truncate or rotate the log file in case the file size grows over a certain limit. So make sure you configure log rotation on your server.

7.4 Using the Web Feature Service

The Web Feature Service is implemented against version 2.0 of the OGC Web Feature Service Interface Standard. Previous versions are not supported any more, and clients have to make sure to use this version of the interface when sending requests to the WFS service.

The following chapters provide a documentation of the functionality offered by the 3D City Database Web Feature Service. They *do not provide* a general overview or description of the

OGC Web Feature Service Interface Standard itself. If you need more general information about WFS, please refer to the WFS specification document instead (OGC Doc. No. 09-025r2).

7.4.1 Basic functionality

7.4.1.1 WFS operations

The OGC WFS 2.0 interface defines eleven operations that can be invoked by a client. A WFS server is not required to offer all operations in order to conform to the standard but may support a subset only. For this purpose, the WFS standard defines *conformance classes* named *Simple WFS*, *Basic WFS*, *Transactional WFS* and *Locking WFS* that grow in the number of mandatory operations. The current version of the 3D City Database Web Feature Service implements the *Simple WFS* conformance class. Thus, it is *fully OGC conformant* but lacks operations from other conformance classes. It is planned to incrementally increase the functionality of the WFS in future releases.

The following table lists all WFS 2.0 operations and marks those supported by the 3D City Database WFS.

Operation	Description	Supported
GetCapabilities	The GetCapabilities operation generates a service metadata document describing the WFS service provided by a server.	X
DescribeFeatureType	The DescribeFeatureType operation returns a schema description of the CityGML feature types offered by the WFS instance.	X
ListStoredQueries	The ListStoredQueries operation lists the stored queries available at the server.	X
DescribeStoredQuery	The DescribeStoredQueries operation provides detailed metadata about each stored query expression that the server offers.	X
GetFeature	The GetFeature operation returns a selection of CityGML features from the 3D City Database using a query expression.	X
GetPropertyValue	The GetPropertyValue operation allows the value of a feature property or part of the value of a complex feature property to be retrieved from the 3D City Database for a set of features identified using a query expression.	-
LockFeature	The LockFeature operation is used to expose a long-term feature locking mechanism to ensure consistency in data manipulation operations (e.g., update or delete).	-
GetFeatureWithLock	The GetFeatureWithLock operation is functionally similar to the GetFeature operation except that in response to a GetFeatureWithLock operation, the WFS shall also lock the features in the result set.	-
CreateStoredQuery	A stored query may be created using the CreateStoredQuery operation.	-
DropStoredQuery	The DropStoredQuery operation allows previously created stored queries to be dropped from the system.	-
Transaction	The Transaction operation is used to describe data transformation operations (i.e., insert, update, replace, delete) to be applied to CityGML feature instances under the control of the web feature service.	-

Table 27: Overview of supported WFS 2.0 operations.

7.4.1.2 Service URL

The *service URL* or service endpoint is the location where the 3D City Database WFS can be accessed by a client application over a local network or the internet. This URL is typically composed as follows:

```
http[s]://[host][:port]/[context_path]/wfs
```

The actual URL depends on the servlet container and your network configuration. Please ask your network administrator for the *protocol* (typically `http` or `https`), the *host* name and the *port* of the server. The *context path* is typically added to the URL by the servlet container. Please refer to the documentation of your servlet container for more information. The last component `wfs` of the URL identifies the service and makes sure that requests are routed to the WFS service implementation.

Note: For Apache Tomcat, the name of the WFS WAR file will be used as *context path* in the service URL. For example, if the WAR file is named `citydb-wfs.war`, then the service URL will be `http[s]://[host][:port]/citydb-wfs/wfs`. To pick a different context path, simply rename the WAR file or change Tomcat's default behavior.

7.4.1.3 Service bindings

A *service binding* refers to the communication protocol that shall be used for exchanging request and response messages between a WFS server and a client. The WFS 2.0 interface standard defines *HTTP GET*, *HTTP POST* and *SOAP over HTTP POST* as possible service bindings for WFS 2.0 implementations.

The 3D City Database WFS implements the *HTTP POST* conformance class only. Therefore, **all client requests** must be encoded in XML and sent to the server using the HTTP method *POST* (using `text/xml` as content type). The *HTTP GET* method and key-value pair (KVP) encoded WFS requests are *not supported*. The only exception from this rule is the `GetCapabilities` operation for which the WFS service will also answer a KVP-encoded HTTP GET request. Note that the XML content of POST messages sent to the server must be well-formed and valid with respect to the WFS 2.0 XML Schema.³

The following table summarizes the operations and the supported service binding as offered by the 3D City Database WFS.

Operation	Service Binding
GetCapabilities	XML over HTTP POST and KVP over HTTP GET
DescribeFeatureType	XML over HTTP POST
ListStoredQueries	XML over HTTP POST
DescribeStoredQuery	XML over HTTP POST

Table 28: Service bindings for the supported WFS 2.0 operations.

³ <http://schemas.opengis.net/wfs/2.0/wfs.xsd>

7.4.1.4 CityGML feature types

The 3D City Database WFS supports all CityGML *top-level feature types*, and corresponding feature instances will be sent to the client upon request. If you want to only advertise a subset of the CityGML feature types to clients though, you can restrict the feature types in the `config.xml` settings (cf. chapter 7.3.3).

Note: *Appearance* properties of CityGML features such as textures or color information are *currently not supported* by the WFS implementation and thus will not be included in a response document.

The supported CityGML feature types together with their official XML namespaces (CityGML version 2.0 and 1.0) are listed in the table below.

Operation	Service Binding
Building	http://www.opengis.net/citygml/building/2.0 http://www.opengis.net/citygml/building/1.0
TransportationComplex	http://www.opengis.net/citygml/transportation/2.0 http://www.opengis.net/citygml/transportation/1.0
Road	http://www.opengis.net/citygml/transportation/2.0 http://www.opengis.net/citygml/transportation/1.0
Track	http://www.opengis.net/citygml/transportation/2.0 http://www.opengis.net/citygml/transportation/1.0
Square	http://www.opengis.net/citygml/transportation/2.0 http://www.opengis.net/citygml/transportation/1.0
Railway	http://www.opengis.net/citygml/transportation/2.0 http://www.opengis.net/citygml/transportation/1.0
CityFurniture	http://www.opengis.net/citygml/cityfurniture/2.0 http://www.opengis.net/citygml/cityfurniture/1.0
LandUse	http://www.opengis.net/citygml/landuse/2.0 http://www.opengis.net/citygml/landuse/1.0
WaterBody	http://www.opengis.net/citygml/waterbody/2.0 http://www.opengis.net/citygml/waterbody/1.0
PlantCover	http://www.opengis.net/citygml/vegetation/2.0 http://www.opengis.net/citygml/vegetation/1.0
SolitaryVegetationObject	http://www.opengis.net/citygml/vegetation/2.0 http://www.opengis.net/citygml/vegetation/1.0
ReliefFeature	http://www.opengis.net/citygml/relief/2.0 http://www.opengis.net/citygml/relief/1.0
GenericCityObject	http://www.opengis.net/citygml/generics/2.0 http://www.opengis.net/citygml/generics/1.0
CityObjectGroup	http://www.opengis.net/citygml/cityobjectgroup/2.0 http://www.opengis.net/citygml/cityobjectgroup/1.0

Table 29: Supported CityGML top-level feature types together with their XML namespace.

7.4.1.5 Exception reports

In the event that the WFS encounters an error while parsing or processing a request, an XML document indicating that error is generated and sent to the client as exception response. Please refer to the WFS 2.0 specification for the structure and syntax of the exception response.

7.4.2 GetCapabilities operation

The `GetCapabilities` operation generates an XML-encoded service metadata document describing the WFS service provided by a server. The *capabilities* document contains relevant

technical and non-technical information about the service and its provider. Its content mainly depends on the configuration of the WFS in the `config.xml` settings file (if created dynamically).

The following XML snippet shows an XML encoding of a `GetCapabilities` operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetCapabilities service="WFS" xmlns:wfs="http://www.opengis.net/wfs/2.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/wfs/2.0
  http://schemas.opengis.net/wfs/2.0/wfs.xsd"/>
```

Listing 10: Example `GetCapabilities` operation.

The declaration of the WFS XML namespace `http://www.opengis.net/wfs/2.0` is mandatory to be able to validate the request against the official WFS XML Schema definition. The reference to the schema location using the `xsi:schemaLocation` attribute is however optional. It is *recommended* though if the XML encoding of the request is created manually by the user (and not automatically by a client software) in order to ensure schema validity. Per default, the WFS service will reject invalid requests (see chapter 7.3.4).

The following table shows the XML attributes that can be used in the `GetCapabilities` request and are supported by the WFS implementation.

XML attribute	O / M _a	Default value	Description
service	M	WFS (fixed)	The service attribute indicates the service type. The value "WFS" is fixed.
^a O = optional, M = mandatory			

Listing 11: Supported XML attributes of a `GetCapabilities` operation.

As alternative to XML encoding, the `GetCapabilities` operation may also be invoked through a KVP-encoded HTTP GET request.

```
http[s]://[host][:port]/[context_path]/wfs?request=
GetCapabilities&service=WFS&AcceptVersions=2.0.0,2.0.2
```

The `service` parameter is also mandatory for the KVP-encoded request. In addition, the `AcceptVersions` parameter can be used for version number negotiation with the WFS server (cf. OGC Document No. 06-121r3:2009, chapter 7.3.2).

7.4.3 DescribeFeatureType operation

The `DescribeFeatureType` operation returns an XML Schema description of the CityGML feature types advertised by the 3D City Database WFS instance. Which feature types are offered by the WFS is controlled through the `config.xml` settings file (cf. chapter 7.4.1.4). The XML Schema defines the structure and content of the features (thematic and spatial attributes, nested features, etc.) as well as the way how features are encoded in responses to `GetFeature` requests.

The following example shows a valid `DescribeFeatureType` operation requesting the XML Schema definition of the CityGML 1.0 *Building* feature type.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:DescribeFeatureType service="WFS" version="2.0.0"
  xmlns:wfs="http://www.opengis.net/wfs/2.0"
  xmlns:bldg="http://www.opengis.net/citygml/building/1.0">
  <wfs:TypeName>bldg:Building</wfs:TypeName>
</wfs:DescribeFeatureType>
```

Listing 12: Example `DescribeFeatureType` operation.

The `DescribeFeatureType` operation takes the following XML attributes.

XML attribute	O / M _a	Default value	Description
service	M	WFS (fixed)	The service attribute indicates the service type. The value "WFS" is fixed.
version	M	2.0.x	The version of the WFS Interface Standard to be used in the communication.
outputFormat	O	application/gml+xml; version=3.1	Controls the format of the schema description. Currently, the default value is the only option and results in a CityGML / GML 3.1.1 application schema.
handle	O		The handle parameter allows a client to associate a mnemonic name with the request that will be used in exception reports.
^a O = optional, M = mandatory			

Listing 13: Supported XML attributes of a `DescribeFeatureType` operation.

The `<wfs:TypeName>` child element of the `DescribeFeatureType` operation identifies the feature type for which the XML Schema description is requested. Be careful to use the correct spelling of the feature type name (as specified by the CityGML standard) and to associate the name with the correct CityGML XML namespace. The `<wfs:TypeName>` element may occur multiple times to request schema definitions of several feature types in a single `DescribeFeatureType` operation. If the `<wfs:TypeName>` element is omitted, then the CityGML base schema is returned by the WFS.

7.4.4 ListStoredQueries operation

Since version 2.0 of the WFS standard, a WFS server is supposed to manage predefined and parameterized feature query expressions (so called *stored queries*) that are stored by the server and that can be repeatedly invoked by the client using different parameter values. Stored queries hide the complexity of the underlying query expression from the client since all the client needs to know is the unique identifier of the stored query as well as the names and types of the parameters in order to invoke the operation.

The `ListStoredQuery` operation is meant to provide the list of stored queries that is offered by the WFS server. The response document contains the unique identifier for each stored query which can then be used in a subsequent `DescribeStoredQuery` operation in order to

receive the details of a specific stored query from the WFS server. The following listing presents an example `ListStoredQuery` operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:ListStoredQueries service="WFS" version="2.0.0"
  xmlns:wfs="http://www.opengis.net/wfs/2.0"/>
```

Listing 14: Example `ListStoredQuery` operation.

The `ListStoredQuery` operation may take the following XML attributes as parameters.

XML attribute	O / M _a	Default value	Description
service	M	WFS (fixed)	The service attribute indicates the service type. The value "WFS" is fixed.
version	M	2.0.x	The version of the WFS Interface Standard to be used in the communication.
handle	O		The handle parameter allows a client to associate a mnemonic name with the request that will be used in exception reports.
^a O = optional, M = mandatory			

Listing 15: Supported XML attributes of a `ListStoredQuery` operation.

7.4.5 DescribeStoredQuery operation

The `DescribeStoredQuery` operation is used to provide the details of one or more stored queries offered by the server. The following listing exemplifies a `DescribeStoredQuery` request.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:DescribeStoredQueries service="WFS" version="2.0.0"
  xmlns:wfs="http://www.opengis.net/wfs/2.0">
  <wfs:StoredQueryId>http://www.opengis.net/def/query/OGC-
WFS/0/GetFeatureById</wfs:StoredQueryId>
</wfs:DescribeStoredQueries>
```

Listing 16: Example `DescribeStoredQuery` operation.

The `<wfs:StoredQueryId>` child element provides the unique identifier of the stored query (see `ListStoredQuery` operation, chapter 7.4.4). By providing more than one unique identifier through multiple `<wfs:StoredQueryId>` elements, the descriptions of separate stored queries can be requested in a single `DescribeStoredQuery` operation. If the `<wfs:StoredQueryId>` element is omitted, a description of all stored queries available at the WFS server is returned to the client.

The above request will produce a response similar to the following listing.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wfs:DescribeStoredQueriesResponse xmlns:fes="http://www.opengis.net/fes/2.0"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wfs="http://www.opengis.net/wfs/2.0">
  <wfs:StoredQueryDescription id="http://www.opengis.net/def/query/OGC-
WFS/0/GetFeatureById">
    <wfs:Title xml:lang="en">Get feature by identifier</wfs:Title>
```

```

<wfs:Abstract xml:lang="en">Retrieves a feature by its gml:id.</wfs:Abstract>
<wfs:Parameter name="id" type="xs:string">
  <wfs:Title xml:lang="en">Identifier</wfs:Title>
  <wfs:Abstract xml:lang="en">The gml:id of the feature to be retrieved.</wfs:Abstract>
</wfs:Parameter>
<wfs:QueryExpressionText returnFeatureTypes="" language="urn:ogc:def:queryLanguage:OGC-
WFS::WFS_QueryExpression" isPrivate="false">
  <wfs:Query typeNames="schema-element(core:_CityObject)">
    <fes:Filter>
      <fes:ResourceId rid="{id}"/>
    </fes:Filter>
  </wfs:Query>
</wfs:QueryExpressionText>
</wfs:StoredQueryDescription>
</wfs:DescribeStoredQueriesResponse>

```

Listing 17: Example response to a DescribeStoredQuery request.

Every WFS implementation must at minimum offer the `GetFeatureById` stored query having the unique identifier <http://www.opengis.net/def/query/OGC-WFS/0/GetFeatureById> as shown above. This stored query takes a single parameter `id` of type `xs:string` and returns zero or exactly one feature whose resource identifier matches the `id` value. For the 3D City Database WFS, the `id` value is evaluated against the `gml:id` of each feature in the database to find a match.

The `returnFeatureTypes` attribute lists the feature types that may be returned by a stored query. Note that this string is empty for the `GetFeatureById` query. Consequently, the query will return a feature instance of all advertised feature types if its `gml:id` matches. The set of advertised feature types can be influenced in the `config.xml` settings file.

The `DescribeStoredQuery` operation allows the following XML attributes.

XML attribute	O / M ^a	Default value	Description
service	M	WFS (fixed)	The service attribute indicates the service type. The value "WFS" is fixed.
version	M	2.0.x	The version of the WFS Interface Standard to be used in the communication.
handle	O		The handle parameter allows a client to associate a mnemonic name with the request that will be used in exception reports.
^a O = optional, M = mandatory			

Listing 18: Supported XML attributes of a DescribeStoredQuery operation.

7.4.6 GetFeature operation

The `GetFeature` operation lets a client query CityGML features from the 3D City Database. The *Simple WFS* conformance class only mandates WFS server implementations to support `GetFeature` queries that use the predefined stored query `GetFeatureById` as query expression and filter criteria. For this reason, the current version of the 3D City Database WFS

handles `GetFeatureById` queries but no ad-hoc queries. The `GetFeature` support will be extended in future releases.

A valid `GetFeature` operation is shown below. The `gml:id` of the city object that shall be returned by the WFS is passed as parameter to the `GetFeatureById` stored query.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature service="WFS" version="2.0.0" xmlns:wfs="http://www.opengis.net/wfs/2.0">
  <wfs:StoredQuery id="http://www.opengis.net/def/query/OGC-WFS/0/GetFeatureById">
    <wfs:Parameter name="id">ID_0815</wfs:Parameter>
  </wfs:StoredQuery>
</wfs:GetFeature>
```

Listing 19: Example `GetFeature` operation.

The WFS will answer the above request with either the CityGML city object(s) whose `gml:id` value matches `ID_0815` or with an exception report in case no matching city object was found in the 3D City Database.

A single `GetFeature` operation can also be used to request more than one feature.

```
<?xml version="1.0" encoding="UTF-8"?>
<wfs:GetFeature service="WFS" version="2.0.0" xmlns:wfs="http://www.opengis.net/wfs/2.0">
  <wfs:StoredQuery id="urn:ogc:def:query:OGC-WFS::GetFeatureById">
    <wfs:Parameter name="id">first gml:id</wfs:Parameter>
  </wfs:StoredQuery>
  <wfs:StoredQuery id="urn:ogc:def:query:OGC-WFS::GetFeatureById">
    <wfs:Parameter name="id">second gml:id</wfs:Parameter>
  </wfs:StoredQuery>
</wfs:GetFeature>
```

Listing 20: Example `GetFeature` operation requesting for two city objects.

If a `GetFeature` request results in more than one city objects or consists of more than one stored query, the response will be wrapped by one or more `<wfs:FeatureCollection>` elements. Please refer to the WFS 2.0 specification for details on the encoding of the response document.

The `GetFeature` operation can be influenced by the following XML attributes.

XML attribute	O / M _a	Default value	Description
service	M	WFS (fixed)	The service attribute indicates the service type. The value "WFS" is fixed.
version	M	2.0.x	The version of the WFS Interface Standard to be used in the communication.
handle	O		The handle parameter allows a client to associate a mnemonic name with the request that will be used in exception reports.
outputFormat	O	application/gml+xml; version=3.1	Controls the encoding of the response. Per default, the WFS uses CityGML / GML 3.1.1. The outputFormat attribute may also take the value

			"GML3.1+GZIP", in which case the CityGML / GML 3.1.1 encoding is additionally compressed using GZIP.
count	O	unlimited	The maximum number of features to be returned by the WFS service.
resultType	O	results	If the value of the resultType parameter is set to "results" the server generates a response document containing features that satisfy the operation. If set to "hits" the server generates an empty response document indicating the count of the total number of features that the operation would return.
^a O = optional, M = mandatory			

Listing 21: Supported XML attributes of a GetFeature operation.

7.5 Web-based WFS client

The 3D City Database WFS is shipped with a simple web-based client that is mainly meant to test the functionality of the server. The client is automatically installed with the server and is available at the following URL (cf. chapter 7.4.1.2 for details):

```
http[s]://[host][:port]/[context_path]/wfsclient
```

The screenshot below shows the user interface of the client rendered in a standard web browser.

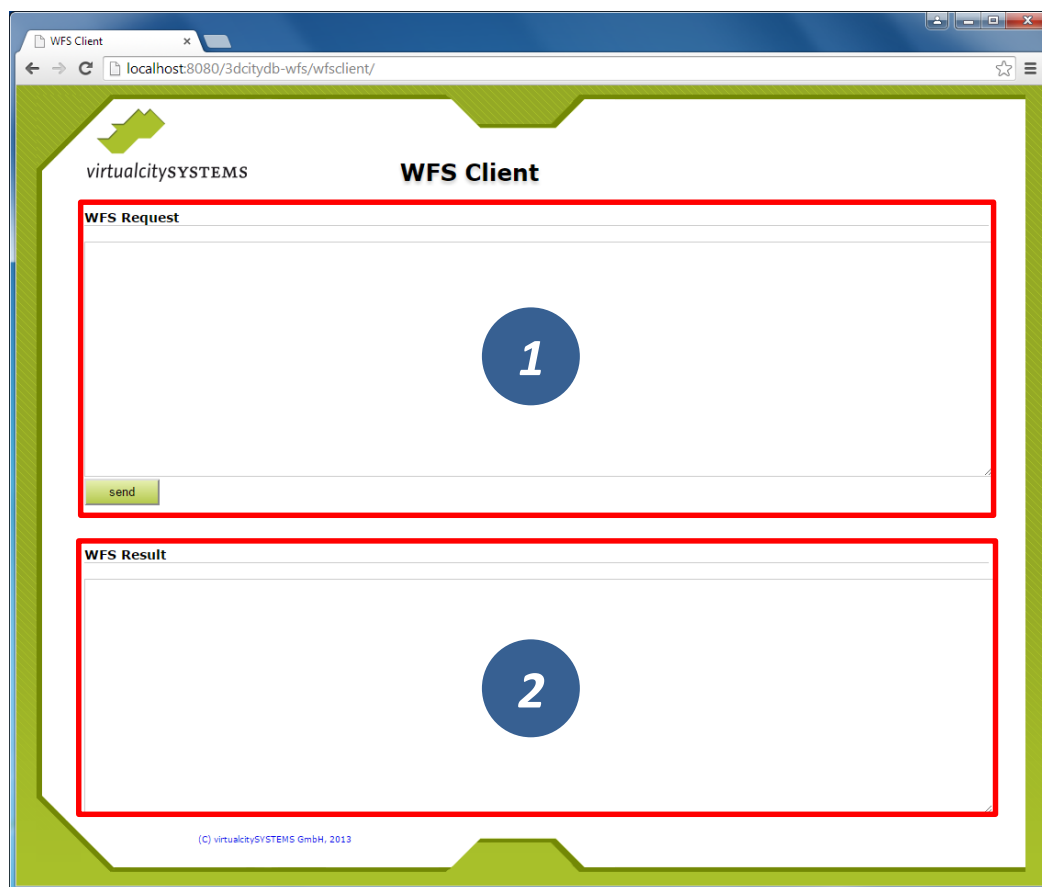


Figure 140: Web-based WFS client.

The user interface consists of two text fields. A user simply enters the XML-encoded operation request that shall be sent to the server into the upper text field named *WFS Request* [1]. Clicking on the *Send* button forwards the request to the server. As soon as the response document is received from the WFS server, it is rendered in the lower text field named *WFS Result*.

Note: Avoid sending requests through this client that might potentially result in a large number of city objects contained in the response document. Otherwise the available main memory of the web browser is quickly exhausted when trying to display the response document, which renders the browser non-responsive or might even lead to a program crash. You may want to use the *count* attribute on the *GetFeature* request in order to limit the maximum number of features to be contained in the response document. Alternatively, you can specify the “*hits*” value for the *resultType* attribute in order to only receive the number of features matching your query instead of the features themselves (cf. chapter 7.4.6).

8 References

3DCityDB Homepage, <http://www.3dcitydb.org/> (accessed March 2015).

Active Server Pages Reference, Microsoft, Weblink (accessed March 2015):

<http://msdn.microsoft.com/en-us/library/ms526064.aspx>

Barners, M., Finch, E. L. (2008): *COLLADA - Digital Asset Schema Release 1.5.0*. The Khronos Group Inc., Sony Computer Entertainment Inc, April 2008.
http://www.khronos.org/files/collada_spec_1_5.pdf

Berlin3D Homepage, Business Location Center Berlin, <http://www.virtual-berlin.de> (accessed March 2015).

Borrmann, A., Kolbe, T. H., Donaubauer, A., Steuer, H., Jubierre, J. R., Flurl, M. (2015): *Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications*. Computer-Aided Civil and Infrastructure Engineering (Volume 30, Issue 4), pp. 263–281

Chaturvedi, K., Yao, Z., Kolbe, T. H. (2015): *Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL*. In: Proc. of the 35th Annual Conference of the German Society for Photogrammetry, Remote Sensing and Geoinformation (DGPF), Weblink (accessed March 2015):
http://www.dgpf.de/src/tagung/jt2015/proceedings/papers/34_DGPF2015_Chaturvedi_et_al.pdf

CityGML Homepage, <http://www.citygml.org> (accessed March 2015).

Coffman, E.G. Jr., Garey, M. R., Johnson, D.S., Tarjan, R.E. (1980): *Performance bounds for level-oriented two-dimensional packing algorithms*. In: SIAM Journal on Computing 9 (1980), pp. 801–826.

Döllner, J., Buchholz, H., Brodersen, F., Glander, T., Jütterschenke, S., Klimetschek, A. (2005): *Smart Buildings – A Concept for Ad-Hoc Creation and Refinement of 3D Building Models*. In: Kolbe, T. H., Gröger, G. (eds.): Proceedings of the 1st International Workshop on Next Generation 3D City Models, Bonn, Germany, June 2005, EuroSDR Publications.

Döllner, J., Kolbe, T. H., Liecke, F., Sgouros, T., Teichmann, K. (2006): *The Virtual 3D City Model of Berlin - Managing, Integrating, and Communicating Complex Urban Information*. In: Proceedings of the 25th Urban Data Management Symposium UDMS 2006 in Aalborg, Denmark, May 15-17.

Foley, J., van Dam, A., Feiner, S., Hughes, J. (1995): *Computer Graphics: Principles and Practice*. Addison Wesley, 2nd Ed.

Google Elevation API, <https://developers.google.com/maps/documentation/elevation/> (accessed March 2015).

- Gröger, G., Kolbe, T. H., Schmittwilken, J., Stroh, V., Plümer, L. (2005): *Integrating versions, history and levels-of-detail within a 3D geodatabase*. In: Kolbe, T. H., Gröger, G. (eds.): Proceedings of the 1st International Workshop on Next Generation 3D City Models, Bonn, Germany, June 2005, EuroSDR Publications.
- Gröger G., Kolbe, T. H., Czerwinski, A., Nagel C. (2008): *OpenGIS® City Geography Markup Language (CityGML) Encoding Standard*, Version 1.0.0. Open Geospatial Consortium, Doc. No. 08-007r1, August 20th.
http://portal.opengeospatial.org/files/?artifact_id=28802
- Gröger G., Kolbe, T. H., Nagel C., Häfele, K. H. (2012): *OpenGIS® City Geography Markup Language (CityGML) Encoding Standard*, Version 2.0.0. Open Geospatial Consortium, Doc. No. 12-019,
http://portal.opengeospatial.org/files/?artifact_id=28802
- Herreruela, J., Nagel, C., Kolbe, T. H. (2012): *Value-added Services for 3D City Models using Cloud Computing*. In: Löwner, M.-O., Hillen, F., Wohlfahrt, R. (eds.): Geoinformatik 2012 "Mobilität und Umwelt", Proc. of the Conference Geoinformatik 2012, 28.-30. 3. 2012 in Braunschweig
- Herring, J. (2001): *The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema)*. OGC Document Number 01-101
- Java Application Launcher (2015): Oracle, Weblink (accessed March 2015):
<http://docs.oracle.com/javase/6/docs/technotes/tools/windows/java.html>
- Kaden, R., Kolbe, T. H. (2014): *Simulation-Based Total Energy Demand Estimation of Buildings using Semantic 3D City Models*. International Journal of 3-D Information Modeling, 3(2), 35-53, April-June 2014
- Kolbe, T. H., Gröger, G. (2003): *Towards unified 3D city models*. In Schiewe, J., Hahn, M., Madden, M., Sester, M. (eds.): Proceedings of the ISPRS Comm. IV Joint Workshop on Challenges in Geospatial Analysis, Integration and Visualization II in Stuttgart
- Kolbe, T. H. (2009): *Representing and Exchanging 3D City Models with CityGML*. In: Lee, J., Zlatanova, S. (eds.): Proceedings of the 3rd International Workshop on 3D Geo-Information 2008 in Seoul, South Korea. Lecture Notes in Geoinformation & Cartography, Springer Verlag, 2009.
- Kolbe, T. H.; König, G.; Nagel, C.; Stadler, A. (2009): *3D-Geo-Database for CityGML*, Documentation Version 2.0.1, Institute for Geodesy and Geoinformation Science, TU Berlin. Weblink (accessed March 2015):
http://www.3dcitydb.de/3dcitydb/fileadmin/downloaddata/3DCityDB-Documentation-v2_0.pdf
- Kunde, F. (2013): *CityGML in PostGIS: portability, usage and performance analysis using the example of the 3D City Database of Berlin*. (in german only) Master Thesis, University of Potsdam, Germany, URN: urn:nbn:de:kobv:517-opus-63656 (accessed March 2015).

- Lodi A., Martello S., Vigo D. (1999): *The Touching Perimeter Algorithm: Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems*. In: INFORMS J on Computing: pp. 345-357.
- Lodi A., Martello S., Monaci M., (2002): *Two-dimensional packing problems: A survey*. In: European Journal of Operational Research, 141, issue 2, pp. 241-252.
- Murray, C. et al. (2010): *Oracle® Spatial Developer's Guide 11g Release 2 (11.2)*, E11830-06, March 2010.
http://docs.oracle.com/cd/E18283_01/appdev.112/e11830.pdf
- Nagel, C., Stadler, A. (2008): *Die Oracle-Schnittstelle des Berliner 3D-Stadtmodells*. In: Clemen, C. (Ed.): *Entwicklerforum Geoinformationstechnik 2008*, Shaker Verlag, Aachen, S. 197-221.
- Plümer, L., Gröger, G., Kolbe, T. H., Schmittwilken, J., Stroh, V., Poth, A., Taddeo, U. (2005): *3D-Geodatenbank Berlin, Dokumentation V1.0* Institut für Kartographie und Geoinformation der Universität Bonn (IKG), lat/lon GmbH. Weblink <http://www.ikg.uni-bonn.de/index.php?id=253> (accessed March 2015).
- Stadler, A., Nagel, C., König, G., Kolbe, T. H. (2009): *Making interoperability persistent: A 3D geo database based on CityGML*. In: Lee, J., Zlatanova, S. (eds.): *Proceedings of the 3rd International Workshop on 3D Geo-Information 2008 in Seoul, South Korea*. Lecture Notes in Geoinformation & Cartography, Springer Verlag, 2009.
- Whiteside, A. (2009): *Definition identifier URNs in OGC namespace*, Version 1.3. Open Geospatial Consortium, OGC® Best Practices, Doc. No. 07-092r3, January 15th.
http://portal.opengeospatial.org/files/?artifact_id=30575
- Wilson, T. (2008): *OGC® KML*, OGC® Standard Version 2.2.0. Open Geospatial Consortium, Doc. No. 07-147r2, April 14th.
http://portal.opengeospatial.org/files/?artifact_id=27810
- Weisstein, E. W. (2015): *Affine Transformation*, Wolfram MathWorld, Weblink (accessed March 2015): <http://mathworld.wolfram.com/AffineTransformation.html>
- Yao, Z., Sindram, M., Kaden, R., Kolbe, T. H. (2014): *Cloud-basierter 3D-Webclient zur kollaborativen Planung energetischer Maßnahmen am Beispiel von Berlin und London*. In: Kolbe, T. H., Bill, R., Donaubauer, A. (eds.): *Geoinformationssysteme 2014 – Beiträge zur 1. Münchner GI-Runde*, 24.-25. 2. 2014, Wichmann Verlag, Heidelberg.

Appendix A Changelog

This appendix provides an overview of the most important changes in version 3.0 of the 3D City Database and the Importer/Exporter compared to the previous release version.

A.1 3D City Database relational schema

A.1.1 General changes

- PostgreSQL: all tables are created in a separate schema called 'citydb' by default; the 'public' schema is not used anymore.
- Renamed most foreign key constraints in order to directly see which integrity is affected when violated.
- Columns 'name', 'name_codespace' and 'description' removed from object tables and added to cityobject table instead.
- Added columns 'class', 'function' and 'usage' to object tables .
- Additional attributes of attributes (e.g. 'codespace' of the 'function' attribute) are mapped to additional columns (using '_codespace' or '_unit' suffixes).
- Tables that store more than one CityGML feature type use an 'objectclass_id' column to identify the feature type (not a string- or flag-based classification like in version 2.x).
- Tables of CityGML features that can be spatially represented with both a gml:MultiSurface and a gml:Solid now contain corresponding columns called lod[x]_multi_surface_id and lod[x]_solid_id.
- Tables of CityGML features that can be spatially represented with an arbitrary geometry (i.e., a subtype of gml:_Geometry) now contain a column called 'lod[x]_brep_id' for surface-based representations and a column called 'lod[x]_other_geom' for other representations (e.g., point-based or line-based representation).

A.1.2 IMPLICIT_GEOMETRY & SURFACE_GEOMETRY

- According to the new naming scheme for geometry columns, the 'implicit_geometry' table now contains the columns 'relative_brep_id' and 'relative_other_geom'.
- Implicit geometries are now stored in an additional column called 'implicit_geometry' in the 'surface_geometry' table in order to not affect the spatial index on the 'geometry' column. As implicit geometries are typically given in a local coordinate system, the 'implicit_geometry' column has no SRID assigned to it.
- The new column 'solid_geometry' stores the real 3D solid of a city object by using a 3D geometry data type of the underlying spatial DBMS. The solids are only stored on the level of the root entries of a solid geometry (is_solid = 1) and are just an addition to the decomposition into separate surfaces, not a replacement.
- Entries in 'surface_geometry' now reference directly to their city object through the column 'cityobject_id'. For implicit geometries, the value of this column is NULL since implicit geometries are typically reused by more than one city object.

A.1.3 BUILDING_INSTALLATION & OPENING

- With CityGML 2.0 it is possible to use prototype objects also for building installations and openings. Therefore, the corresponding tables now contain the columns lod[x]_implicit_rep_id, lod[x]_implicit_ref_pt and lod[x]_implicit_transformation.
- Another addition are boundary surfaces for building installations. This relation is reflected by the 'building_installation_id' column in the 'thematic_surface' table.

A.1.4 New tables for new thematic modules BRIDGE & TUNNEL

- The new thematic modules 'Bridge' and 'Tunnel' introduced in CityGML 2.0 are mapped to corresponding tables. Please, refer to chapter 2 for more details.

A.1.5 CITYOBJECT

- Column 'class_id' is now called 'objectclass_id'.
- Data types of date attributes were changed to TIMESTAMP WITH TIME ZONE.
- New columns 'relative_to_terrain' and 'relative_to_water' were added.

A.1.6 SOLITARY_VEGETAT_OBJECT

- Fixed typo in column 'trunk_diameter' (was 'trunc_diameter').

A.1.7 RASTER_RELIEF, GRID_COVERAGE & ORTHOPHOTO

- Table 'ratser_relief' is now linked to the table 'relief_component' and not 'relief', which was dropped.
- The new table 'grid_coverage' is introduced as a global table where raster data about city objects can be registered (similar to the concept of the table 'surface_geometry'). At the moment there is only one table pointing to 'grid_coverage', which is 'raster_relief' (through the 'grid_coverage_id' column).
- Oracle: There is an additional table called 'grid_coverage_rdt' to store raster data using the data type 'sdo_raster'.
- The 'orthophoto' table was removed.

A.1.8 SURFACE_DATA & TEX_IMAGE & TEXTUREPARAM

- Textures are stored in an additional table called 'tex_image'. Multiple 'surface_data' entries may reference the same texture through the 'tex_image_id' column, which has been introduced to efficiently deal with texture atlases.
- Oracle: data type for storing textures changed from 'ORDImage' to 'BLOB'.
- Texture coordinates are now stored as a 2D polygon and not as a string anymore.

A.2 3D City Database scripts

- New MIGRATION scripts to upgrade from a 3DCityDB v2.1.0 to v3.0.0.
- PostgreSQL: several changes in CREATE_DB script
 - PostgreSQL version is checked in the beginning to avoid subsequent errors.
 - Reference system for the database is set at the end of the script, not at the beginning.

- The search path of the database is set to include the schemas 'citydb,citydb_pkg,public' (this is reset in the DROP_DB script).
- PostgreSQL: only one script is contained in the SCHEMA folder. This script is automatically generated using pgModeler.
- Oracle: the SQL script from the JDeveloper project is manually divided into four SQL files (CONSTRAINTS, INDEXES, SEQUENCES, and TABLES).
- Oracle: removed PLANNING_MANAGER functionality (tables and scripts).
- Oracle: better support for Oracle Locator.
 - When setting up a 3D City Database instance the user can specify which Oracle licence model he is using for his server (Spatial or Locator)
 - Realized through two different versions for the scripts DROP_DB, ENABLE/DISABLE_VERSIONING, DELETE
- Removed scripts for creating read-only users due to several issues (will be added again in a previous release)

A.3 3D City Database stored procedures

A.3.1 General changes

- Renamed 'GEODB_PKG' to 'CITYDB_PKG' (Oracle: packages start with a 'CITYDB_' prefix).
- Removed scripts of Matching/Merging tool (folders MATCH and MERGE).
- PostgreSQL: removed prefixes in function names to have identical names compared to the Oracle version.
- Nearly all functions and procedures offer the name of the schema to be used (Oracle: user schema, PostgreSQL: database schema) as input parameter. Thus, functions in the CITYDB_PKG can be executed against any given schema. If the parameter is not specified, a default value is used (Oracle: USER, PostgreSQL: 'citydb').

A.3.2 UTIL package

- Oracle: added new data type ID_ARRAY for lists of numbers, and DB_VERSION_OBJ for version information.
- Functions and procedures dealing with coordinate reference systems are now contained in file SRS.sql.
- Removed 'error_msg' function.
- Added new function 'citydb_version' that returns version information about the installed 3DCityDB instance.
- Added new function 'get_seq_values' that returns the next *n* values of a given sequence.
- Added new function 'objectclass_id_to_table_name' that converts an 'objectclass_id' of a city object into the corresponding table name.
- Oracle: new functions 'sdo2geojson3d' and 'get_id_array_size'.
- Added new procedure 'update_table_constraint' that modifies foreign key parameters (ON DELETE CASCADE or ON DELETE RESTRICT).
- Added new procedure 'update_schema_constraints' that calls 'update_table_constraint' on every constraint in a given schema.

- Oracle: new functions 'construct_solid', 'sdo2geojson3d' and 'get_id_array_size'.

A.3.3 IDX package

- Oracle: replaced INDEX_TABLE data type by a table called 'index_table' (similar to the PostgreSQL version)

A.3.4 SRS package

- Introduced a new SRS package that contains all database functions and procedures dealing with coordinate reference systems.
- Updated procedure 'change_db_srid' (and 'change_column_srid') and renamed it to 'change_schema_srid'. New optional parameter 'transform' that allows a user to optionally run a reprojection on the coordinates or just change the SRID metadata.
- PostgreSQL: 'check_srid' function is now part of SRS package.

A.3.5 STAT package

- Function 'table_contents' rewritten in a more generic way.

A.3.6 DELETE package

- Delete operations now return the ID(s) of the deleted feature(s).
- Added new function 'delete_cityobject_cascade' (can only be used if foreign key constraints are set to 'on delete cascade').
- Added new function 'cleanup_schema' (PostgreSQL: uses TRUNCATE CASCADE, Oracle: setting constraints to 'on delete cascade' is necessary).
- PostgreSQL: removed pre_ and post_ functions to just have a small number of top-level delete functions (because there is no concept of private functions like in Oracle).

A.3.7 DELETE_BY_LINEAGE package

- Delete operations now return the ID(s) of the deleted feature(s).

A.4 3D City Database Importer/Exporter

The new version 3.0 of the Importer/Exporter contains many bug fixes as well as stability and performance improvements. A full list of fixes and changes is available from the GitHub repository at <https://github.com/3dcitydb/importer-exporter>.

A.4.1 General changes

- The Importer/Exporter can now connect to both Oracle and PostgreSQL.
- Temporary information required during data imports and exports (e.g., for resolving of XLink references) can now optionally be stored to a local file-based database instead of using temporary tables in the 3D City Database instance.
- The location of the main config file ('project.xml') has been changed to %HOMEDRIVE%%HOMEPATH%\3dcitydb\importer-exporter-3.0\config (Windows 7 and higher) respectively \$HOME/3dcitydb/importer-exporter-3.0/config (UNIX/Linux, Mac OS families). Old config files can still be loaded manually.

A.4.2 CityGML import

- CityGML import now supports CityGML versions 2.0, 1.0 and 0.4.
- A new import log optionally tracks all successfully imported top-level city objects in a separate CSV file. In case an import process aborts abnormally, this file can be used to understand which city objects have been processed and stored in the database before termination.
- The import process now follows a fail-on-first-error strategy, i.e. the import terminates upon the first error thrown instead of trying to continue.
- Improved import of texture atlases. Each texture atlas is only stored once in the database (new table 'tex_image') even if it is referenced by more than one city object.
- Local appearance information is now resolved in main memory to reduce import times instead of using temporary database tables.

A.4.3 CityGML export

- Database content can now be exported to CityGML 2.0 or 1.0. When exporting to CityGML 1.0, feature types only available in CityGML 2.0 such as bridges and tunnels are omitted.
- City object group members can now be exported as-reference (using XLink references) instead of as-value to reduce export times. However, note that filter criteria are not applied in this case, which might result in CityGML files containing non-resolvable XLink references.
- When exporting city objects with textures, the texture image files can now be organized into subfolders. This reduces the number of files per folder.

A.4.4 KML/COLLADA export

- Support for new feature types such as bridges and tunnels which is available in CityGML 2.0.
- Solved bug with big coordinate values (e.g., UTM coordinates) in COLLADA exports.
- Added new visualization styles for point-based and line-based generic city objects.
- Added new and fast algorithm for creating texture atlases (replaces former BASIC algorithm); removed SLEA algorithm.
- Reworked tiling implementation: the number of exported objects is now stable for all tiling strategies ('no tiling', 'automatic tiling' and 'manual tiling').

A.5 Web Feature Service

- New feature: Added a basic Web Feature Service interface for the 3D City Database

Appendix B 3DCityDB @ TU München

The Chair of Geoinformatics⁴ at Technische Universität München (TUM) took over the further development of the 3D City Database from TU Berlin (TUB) when Prof. Kolbe moved from TUB to TUM in 2012. 3DCityDB is being used at TUM in teaching courses on spatial databases and 3D city modeling, in student projects and master theses, and in many past and ongoing research projects.

B.1 Interactive Cloud-based 3D Webclient

We have developed a cloud-based interactive 3D web client. While the previous versions require the Google Earth plugin, the new version is using the Cesium Virtual Globe⁵ framework which is based on HTML5 and WebGL and is running in all modern browsers and on different operating systems. The 3D Webclient links 3D visualization models exported in KML/COLLADA from 3DCityDB with table data exported using the 3DCityDB Spreadsheet Generator and allows viewing, editing, and querying objects and their thematic data [Herreruela et al. 2012; Yao et al. 2014; Chaturvedi et al. 2015]. The following image shows a screenshot of a tool created by TUM for the Energy Atlas Berlin. It estimates building energy demands and allows to interactively explore retrofitting potentials for single or sets of buildings [Kaden & Kolbe 2014]. Thematic data are stored in Google Spreadsheets, where spreadsheet formulas are employed to implement ad-hoc computation of energy values.

3DCityDB Webclient V0.12 ©2012-2015 Chair of Geoinformatics TU München

Fieldname	Value
RoofOrCeilingInsulation	17
ESTIMATED_	HEAT_DEMAND_...
Jan_kwh	13853
Feb_kwh	11661
Mar_kwh	9168
Apr_kwh	4337
May_kwh	1380
Jun_kwh	313
Jul_kwh	0
Aug_kwh	4
Sep_kwh	1466
Oct_kwh	5235
Nov_kwh	10750
Dec_kwh	14520
Year_kwh	72693
Year_m2_kwh	55
REFURBISHMENT_	COSTS_
WallRetrofit	88008
WindowReplacement	179865
GroundFloorRetrofit	6592

Object Selection

Clear Selected Objects Aggregation Appearance

Object ID

BLDG_0003000e007044ee

Count of the selected Objects: 1

⁴ <http://www.gis.bgu.tum.de>

⁵ See <https://www.youtube.com/watch?v=n6xQCIF1MxI> for a demo video and also <http://cesiumjs.org/>

B.2 Research Projects in which 3DCityDB is being used

Semantic 3D city modeling, city system modeling, and indoor navigation are major research fields of the Chair of Geoinformatics at TUM. We have been and are driving the international development of CityGML, IndoorGML, and 3D Portrayal web services within the OGC. We are partners in and/or coordinators of projects on Smart Cities, Sustainable Urban Development, and Strategic Energy Planning funded by the [Climate-KIC](#) of the European Institute of Innovation & Technology (EIT). Projects using 3DCityDB are: [Energy Atlas Berlin](#)⁶, Neighborhood Demonstrators, [Smart Sustainable Districts](#)⁷, and [Modeling City Systems](#)⁸.

3DCityDB is also being employed in the collaborative research project ‘[3D Tracks](#)⁹ - Computer-Aided Collaborative Subway Track Planning in Multi-Scale 3D City and Building Models’ [Borrmann et al. 2015] funded by the German Science Foundation (DFG).

B.3 Current and future work on 3DCityDB

The team at the Chair of Geoinformatics is currently working on the following tools and extensions to 3DCityDB. They will be made available as Open Source software within the 3DCityDB repository as soon as they are finished and tested:

Support of CityGML ADEs: The next big step in the 3DCityDB development will be the support of CityGML Application Domain extensions. We will parse and interpret the XML schema files of the ADE and dynamically create new database tables accordingly. All Importer/Exporter tools will be updated to take into account ADE data.

Spreadsheet Importer: This is the counterpart plugin to the ‘Spreadsheet Generator’. It will allow importing and updating thematic data given in spreadsheets (in CSV or Microsoft Excel format, or Google Spreadsheet) into/within 3DCityDB. Each table row represents data of one city object. Columns can be mapped to predefined CityGML attributes or to generic attributes.

3D Web-Viewer based on Cesium: This is an extension to the Open Source browser-based WebGL 3D viewer application ‘Cesium Virtual Globe’. Our extensions comprise object highlighting, loading and unloading of tiled data, tile prefetching, support of KML network links and regions, and generation of textured 3D models in glTF format.

Solar potential analysis: This tool computes the solar energy of direct and diffuse irradiation on building walls and roofs. The computation considers shadow casting by buildings, the digital surface model and the digital elevation model. The monthly energy and irradiation values as well as the sky view factors are attached as generic attributes to all wall and roof surface objects and in aggregated form to all buildings. Currently, the software is implemented as an FME workbench and a set of database scripts.

⁶ See <http://www.gis.bgu.tum.de/en/projects/energieatlas-berlin/> and <http://energyatlas.energie.tu-berlin.de/>

⁷ <https://www.gis.bgu.tum.de/en/projects/smart-sustainable-districts-ssd/>

⁸ <https://www.gis.bgu.tum.de/en/projects/modeling-city-systems-mcs/>

⁹ <https://www.gis.bgu.tum.de/en/projects/3dtracks/> and <http://www.3dtracks.kit.edu/english/index.php>

Appendix C 3DCityDB @ virtualcitySYSTEMS

virtualcitySYSTEMS¹⁰ has successfully applied the 3D City Database in customer projects worldwide and also funded its development. With the Open Source database at the core, virtualcitySYSTEMS also offers a *3D Spatial Data Infrastructure* solution for the *management, distribution, maintenance* and *visualization* of massive 3D geo data (see next page). As leading developers of the 3D City Database joined the company, virtualcitySYSTEMS now takes an active role in its development. Moreover, virtualcitySYSTEMS offers a branded version of the 3D City database called the *virtualcityDATABASE* to answer customer demands and to provide support and maintenance.

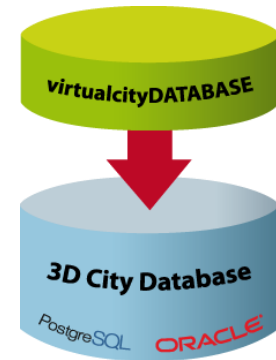


Figure 141:
Extending the 3D
City Database

C.1 virtualcityDATABASE

The virtualcityDATABASE provides enhanced database functionality as well as plugins for the Importer/Exporter tool that support workflows for maintaining and updating the 3D city model content. Main features are:

- Integration of additional LoDs against existing city objects in the database**
 This plugin allows for integrating city objects from an external data source with existing city objects stored in the database. The candidate objects are identified with the database objects based on thematic and spatial checks. Therefore, data inconsistency can easily be spotted and analyzed before an import. If an integration is performed, exiting LoDs are replaced and newly introduced LoDs are attached to the existing objects. Moreover, appearance information can be integrated without replacing the geometry.
- Deletion of entire city objects or single LoDs representations**
 The 3D City Database provides a low-level API for deleting city objects. This API has been extended in the virtualcityDATABASE to also delete single LoDs of city objects. A graphical user dialog realized as a plugin for the Importer/Exporter allows users to easily delete city objects based on comprehensive thematic filter criteria.
- Adding material appearances for buildings**
 This plugin helps to define constant material information for building surfaces based on thematic properties (e.g., to colorize roofs according to their solar potential).
- Transactional Web Feature Service:**
 Customers of the virtualcityDATABASE already benefit from an OGC-compliant WFS 2.0 implementation that supports transactions as well as comprehensive spatial and thematic queries using the OGC Filter Encoding standard.

The virtualcityDATABASE is fully compliant with the 3D City Database. If features developed for the virtualcityDATABASE have gained enough maturity, virtualcitySYSTEMS will introduce them to the Open Source 3D City Database project (e.g. the WFS interface).

¹⁰ <http://www.virtualcitysystems.de/>

C.2 virtualcitySUITE – The 3D City Platform

The virtualcitySUITE is a modular *3D Spatial Data Infrastructure* solution to store, manage, distribute and visualize 3D geo data. Core components are the *virtualcityDATABASE* and its OGC WFS interface for accessing and editing the data, the *virtualcityWAREHOUSE*, a data distribution solution running on FME technology that enables users to export 3D city model content from the *virtualcityDATABASE* into various industry GIS and CAD formats, and the web-based authoring tool *virtualcityPUBLISHER* for creating high-performance 3D web maps. Based on the Open Source 3D City Database, the virtualcitySUITE allows for building a 3D SDI platform for virtual 3D city models based on open standards and interfaces.

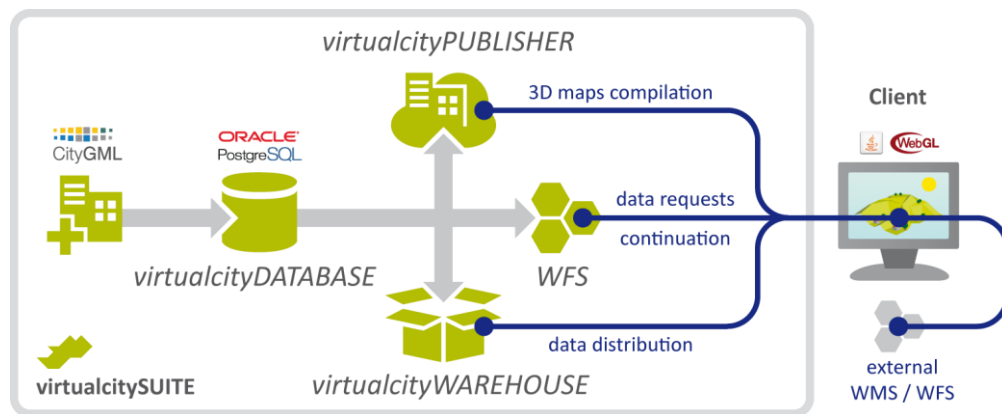


Figure 142: Components of the virtualcitySUITE.

The 3D web maps offer enhanced GIS functionality beyond pure 3D visualization including 3D measurements, real-time shadows, WFS-based thematic and spatial queries, POI integration, routing, data exports through a virtualcityWAREHOUSE interface, and integration of external WMS and WFS data sources as well as oblique and panoramic images.

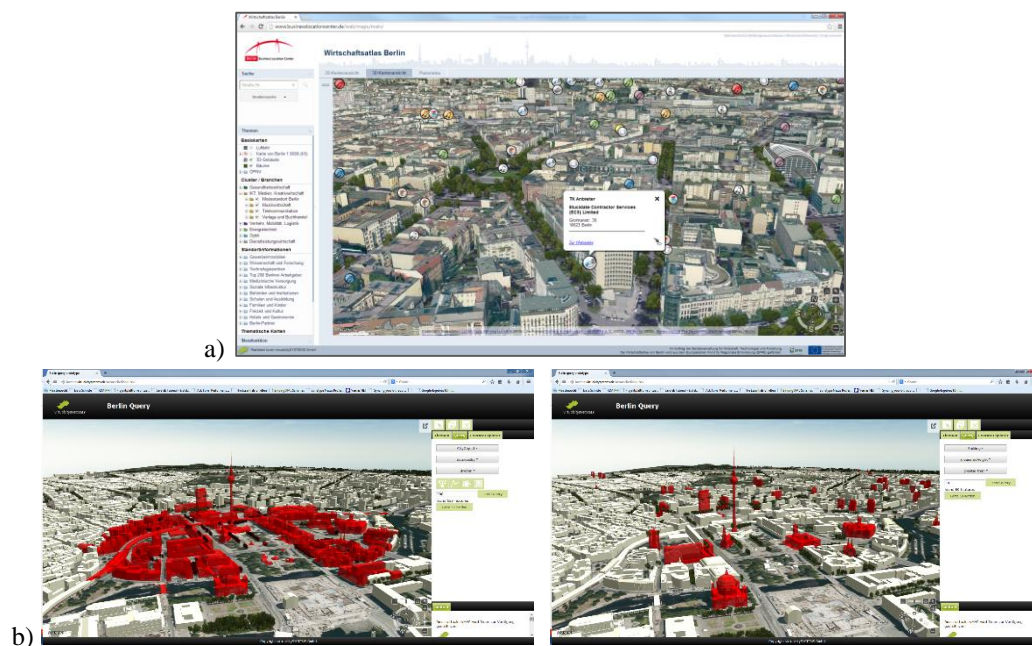


Figure 143: Examples of 3D web maps created from the virtualcityDATABASE: a) The Berlin 3D City Model consisting of 500,000 fully textured buildings, see <http://www.businesslocationcenter.de/wab/maps/main/>, b) WFS-based spatial and thematic selection of city objects in a 3D web map.

Appendix D 3DCityDB @ M.O.S.S.

M.O.S.S. Computer Grafik Systeme GmbH¹¹ is a leading provider of geo topographical data management and processing solutions. Within M.O.S.S. product suite novaFACTORY, the 3D City Database is used since 2011 as the primary storage container for 3D and CityGML based data. M.O.S.S. as an active development partner within the 3D City Database implementation group drives on the technological progress of the 3D City Database. Within the M.O.S.S. customer projects millions of CityGML objects are imported managed and exported by novaFACTORY and the included 3D City Database. One example is the nationwide database for the german LoD1 building product (LOD-DE) which is based on the 3D City Database. novaFACTORY is also used as a 3D platform within different projects concerning renewable energy topics like building heat demand analysis or solar potential assessment.



Figure 144: Example of a 3D building heat demand map for the city of Ludwigsburg created with novaFACTORY 3D within project SimStadt¹²

D.1 novaFACTORY at a glance

novaFACTORY is an advanced Spatial Data Management solution for efficient geodata cataloguing, exploitation and dissemination. With novaFACTORY we are leading the way in the full integration of enterprise-wide geospatial data sources which the whole organization can have access to and work from, covering all aspects of

- **Data Import**
- **Quality Assurance**
- **Data Storage and Management**
- **Data Processing and Enrichment**
- **Data Dissemination**

As applications for geodata have grown, so too has the need to efficiently administer them. Many businesses, whether government departments or private companies, are faced with the complex task of managing geospatial data. The challenge is to allow collaboration across the

¹¹ <http://www.moss.de/>

¹² <http://simstadt.hft-stuttgart.de/>

organization in a meaningful way, from a range of sources and formats located throughout their enterprise.

novaFACTORY is the solution to this challenge. It brings geodata together and eliminates barriers to spatial data usability by automatically uniting disparate data and combining them into one spatial database. novaFACTORY is designed for seamlessly integrating large geographical data sets from many different sources, e.g. topographic maps, digital surface models, aerial photographs or 3D building models.

Within novaFACTORY the module 3D GDI is where the 3D City Database comes into the action.

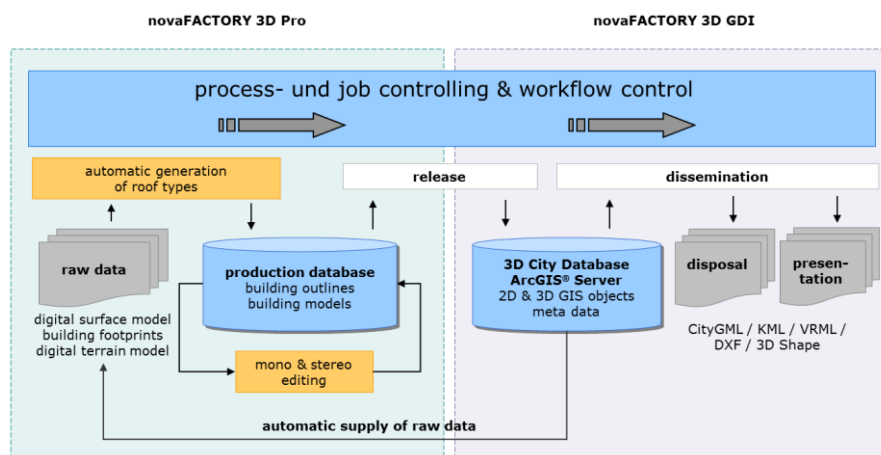


Figure 145: novaFACTORY 3D overview and workflow. 3D data management based on 3D City Database

D.2 novaFACTORY 3D GDI

The novaFACTORY 3D GDI Module is designed for handling and serving 3D city models in CityGML format. It enables the RDBMS based seamless storage and dissemination of 3D city models as well as setting up web services using them. Data is kept within the 3D City Database and can be automatically transferred into an ArcGIS® Geodatabase.

As with all novaFACTORY modules data can be disseminated via an intuitive web interface and via any workstation, in alternatively formats, e.g. CityGML, KML/COLLADA, VRML, 3D Shape, 3D PDF and 3D DXF. Depending on which kind of format is chosen different export parameters can be opted for to show specific object data.

Additional benefit is gained by automatically enhancing the 3D building data. The novaFACTORY 3D GDI Module offers a fully-integrated solar potential analysis during the export, targeted at the area of interest. 3D data can be visualized directly. Appropriate ArcGIS presentation rules will be generated automatically during the export.

The novaFACTORY 3D GDI Module works best in cooperation with the novaFACTORY 3D Pro Module for automatic recognition of building roofs from photogrammetric raw data. This raw data will be supplied automatically and the 3D City Database will be updated automatically when production data are approved.