

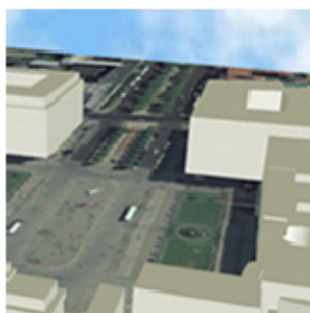
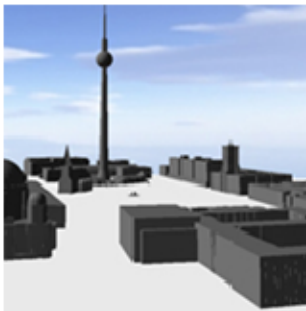
3D City Database for CityGML

3D City Database Version 2.0.5

Importer/Exporter Version 1.3.0

Addendum to the 3D City Database Documentation Version 2.0.1

1 June 2011



Institute for Geodesy and Geoinformation Science
Technische Universität Berlin

Thomas H. Kolbe
Claus Nagel
Javier Herruela

(Page intentionally left blank)

Content

1	DISCLAIMER.....	4
2	OVERVIEW.....	5
3	NEW FEATURES	6
3.1	<i>KML/COLLADA Export.....</i>	6
3.1.1	Main parameters of the KML/COLLADA export.....	7
3.1.2	Preferences.....	11
3.1.2.1	Rendering Preferences.....	11
3.1.2.2	Information Balloon Preferences.....	15
3.1.2.3	Altitude/Terrain Preferences	22
3.1.3	Batch mode	26
3.1.4	Loading exported models in Google Earth.....	27
3.1.5	General setting recommendations	27
3.2	<i>Support for Coordinate Reference Systems (CRS).....</i>	29
3.2.1	General information.....	29
3.2.2	Definition of the CRS for a 3D City Database instance.....	30
3.2.3	Management of user-defined CRSs.....	30
3.2.4	Usage of user-defined CRSs.....	33
3.3	<i>CityGML Export Enhancements.....</i>	35
3.3.1	Coordinate Transformation.....	35
3.3.2	Tiling	36
3.4	<i>Rework and Redesign of the Matching/Merging Tool.....</i>	39
3.5	<i>Extensions to the Database tab and preferences.....</i>	42
3.6	<i>New PL/SQL functionality.....</i>	45
3.6.1	PL/SQL package GEODB_DELETE.....	45
3.6.2	Creating Read-Only Users.....	47
3.7	<i>Test data and template files.....</i>	49
4	REQUIREMENTS	50
5	UPGRADE FROM PREVIOUS VERSIONS OF THE 3D CITY DATABASE ...	51
6	CHANGELOG.....	55
6.1	<i>Changelog for the 3D City Database</i>	55
6.2	<i>Changelog for the Importer/Exporter</i>	56
7	REFERENCES.....	58

1 Disclaimer

The *3D City Database version 2.0.5* and the *Importer/Exporter version 1.3.0* developed by the *Institute for Geodesy and Geoinformation Science (IGG)* at the *Technische Universität Berlin* 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 IGG "AS IS" AND "WITH ALL FAULTS." IGG 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.

IGG 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 IGG.

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

2 Overview

Welcome to the release of the *3D City Database Version 2.0.5* and the *Importer/Exporter Version 1.3.0*. Following almost one year of continuous further development efforts, we are proud to present exciting new features as well as improvements to existing functionality for the 3D City Database and the Importer/Exporter tool.

The most outstanding new feature brought by this release is the **KML/COLLADA export** capability of the Importer/Exporter. 3D City Database contents can now be directly exported in KML and COLLADA formats for presentation, viewing, and visual inspection in a broad range of applications supporting the visualization of KML/COLLADA models, 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.

With this new feature, a link is established between the 3D City Database as means of storage of semantic 3D city models based on CityGML on the one hand, and 3D visualization models based on KML/COLLADA on the other hand which result from a portrayal and styling process of the CityGML base data. Due to the high degree of flexibility and the many configuration options the user can tailor the visualization and the included information assets to the specific needs of different groups of stakeholders.

The following list summarizes the new features and improvements included in this release:

- Support for KML/COLLADA exports (cf. chapter 3.1),
- Support for user-defined Coordinate Reference Systems (cf. chapter 3.2),
- CityGML export enhancements (coordinate transformations, tiling) (cf. chapter 3.3),
- Rework and redesign of the Matching/Merging tool (cf. chapter 3.4),
- Minor changes and extensions (cf. chapters 3.5 and 3.6), and
- Several bug fixes (cf. chapter 6),

The 3D City Database version 2.0.5 is a **mandatory dependency** of the Importer/Exporter version 1.3.0. Thus, **existing 3D City Database instances (version 2.0.3 or below) have to be upgraded** to version 2.0.5 in order to make use of the new features and improvements. An upgrade script is shipped with this release. Please refer to chapter 5 for the documentation of the upgrade procedure.

Please note that this document is an **addendum to the previous documentation of the 3D City Database Version 2.0.1** (cf. [1]). It contains additions and a few corrections of the previous documentation. For a full overview of the 3D City Database and the Importer/Exporter, please also refer to the version 2.0.1 documentation (which is also shipped with the new release).

Further information, software downloads, ready-to-use demos, links to the source code repository, and much more can be found at the **official website of the 3D City Database at <http://www.3dcitydb.net>**.

3 New Features

3.1 KML/COLLADA Export

With this release of the Importer/Exporter, it is possible to export data from a 3D City Database (3DCityDB) instance directly into KML/COLLADA format, ready for visualization in earth browsers such as Google Earth, ArcGIS, and ArcGIS Explorer. Currently only buildings, the most common CityGML feature will be exported. Export of further CityGML features like vegetation, transportation complex, etc. will be accomplished in later releases.

The new *KML/COLLADA Export* tab shown in Fig. 1 collects all parameters required for the export in a similar fashion as for a CityGML export. On the preferences tab a new menu node called *KML/COLLADA Export* containing three subnodes – *Rendering*, *Balloon*, and *Altitude/Terrain* – makes customization of these exports possible.

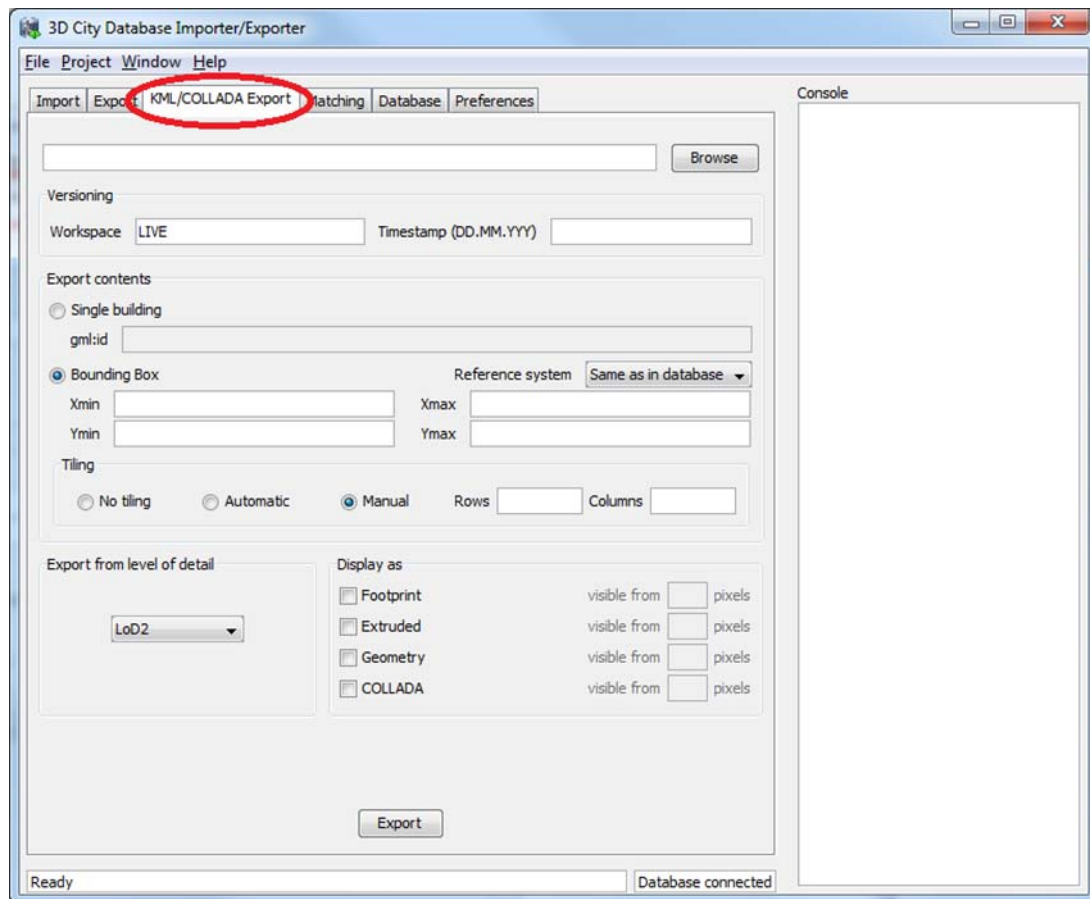


Fig. 1: The new KML/COLLADA Export tab allowing for exporting KML/COLLADA models from the 3DCityDB.

Note: KML/COLLADA formatted exports come straight from the 3D City Database. No direct file transformation CityGML → KML/COLLADA is provided. 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.

3.1.1 Main parameters of the KML/COLLADA export

The input data fields on the *KML/COLLADA Export* tab are from the top down:

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.

Versioning

The Workspace Manager provided by Oracle is a comprehensive tool for version and history management. If the workspace management is activated, it works widely transparent for applications connected to the database. Workspace name and timestamp can be entered here in order to use a certain planning alternative and/or a given point in time as the basis for the KML/COLLADA exports.

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.

Export contents

These fields allow for specifying/selecting the objects of interest for the export. These can be single buildings or whole areas delimited by a bounding box.

- *Single building*: 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. With this release, the possibility to add further user-defined reference systems is introduced (cf. chapter 3.2 for more details). A new reference system can be added to the Import/Export tool (preferences tab, node *Database*, subnode *Reference systems*) as long as they are supported by the Oracle DB server.

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 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.

Untiled exports (*no tiling*) will consist of only one single tile comprising all exported objects, which translates into one single file when output format kmz is chosen. Tiled exports of type *automatic* or *manual* will contain one main file pointing to all tiles. If additionally kmz output is chosen each tile will be exported into one single file. Each tile filename will be enhanced with the tile's row and column number as a suffix. In case a main file is also required for untiled exports (one main file and the actual export), selecting the *manual* option and entering the value 1 for both rows and columns will render that result.

Export from level of detail

The Level of Detail as defined by the CityGML specification [2] which 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.

Display as

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. Pure KML export.

- **COLLADA:** shows the detailed geometry including support for textures. The Appearance/Theme combo box below allows to choose from all possible appearance themes (as defined in the CityGML specification [2]) 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.

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 (1 to 4), Extruded and Geometry require LoD1 or higher, COLLADA exports are possible from LoD2 upwards. Exports will have their filename enhanced with a suffix specifying the selected display form. This applies for both tiled and untiled exports.

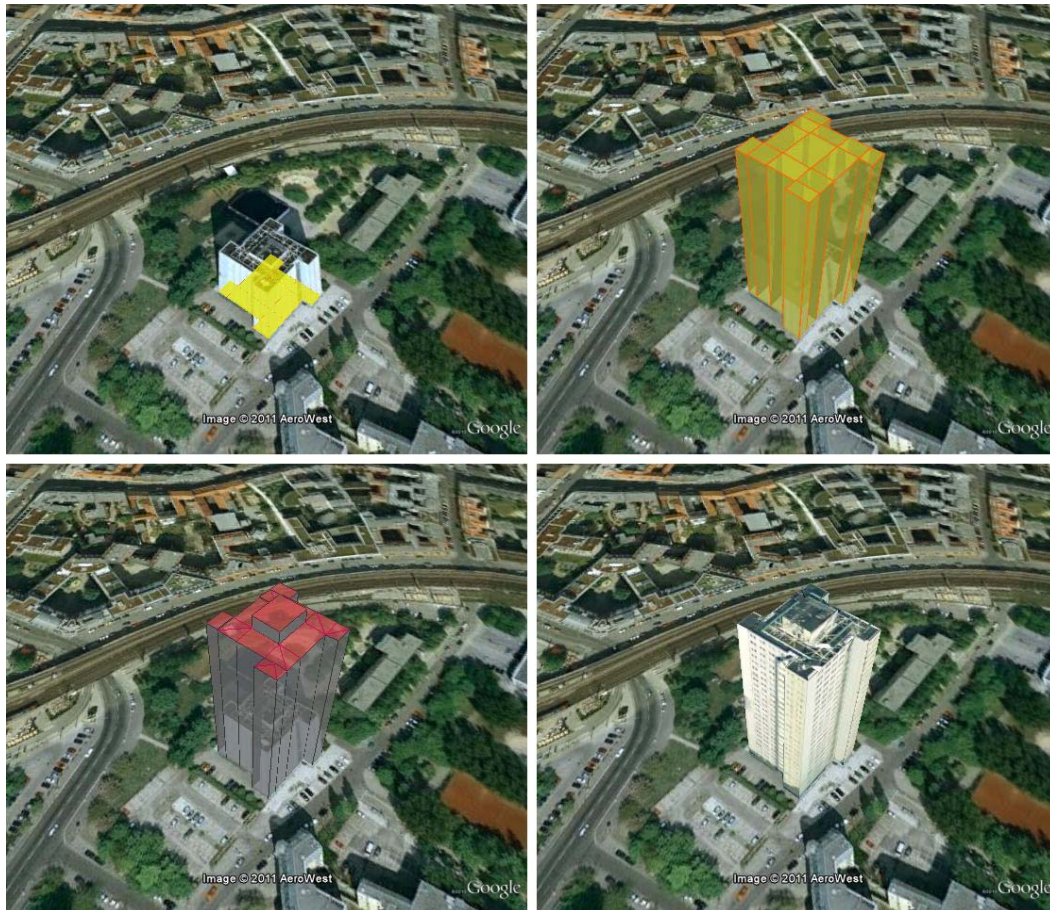


Fig. 2: The same building displayed as (top down and left to right) footprint, extruded, geometry, COLLADA.

With the visibility field next to each display form the user can control the KML element `<minLodPixels>`, see [3]: 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.

3.1.2 Preferences

3.1.2.1 Rendering Preferences

Most aspects regarding the look of the KML/COLLADA exports when visualized in Google Earth can be customized under the preferences tab, node *KML/COLLADA Export*, subnode *Rendering*.

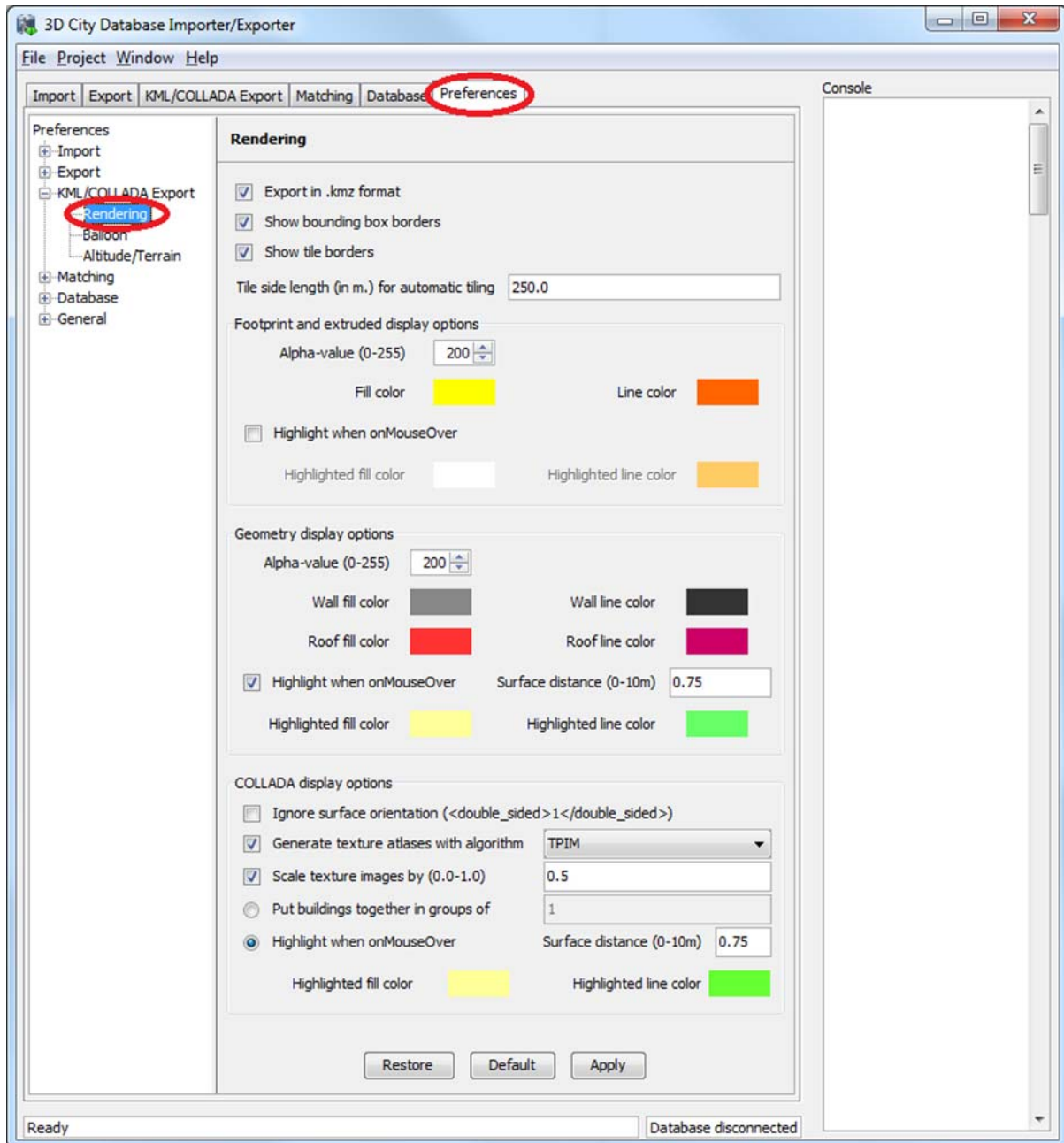


Fig. 3: Rendering settings for the KML/COLLADA export.

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, in a tiled export the main file (pointing to the single tiles) 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.

All remaining settings 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.

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.

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. Whether to group 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 [8] and [12]):

- *BASIC*: the most elementary one. Images are sorted according to decreasing height. Their total width when put next to each other is computed and the square root of this value is taken as the atlas width limit. Texture images are then added left to right following their decreasing heights. When the atlas width limit is reached a new row of images is started within the atlas.
- *SLEA*: Sleater's algorithm (see [11]). Consists of four steps: (1) all items of width greater than 1/2 are packed on top of one another in the bottom of the strip. Suppose h_0 is the height of the resulting packing. All subsequent packing will occur above h_0 . (2) Remaining items are ordered by non-increasing height. A level of items are packed (in non-increasing height order) from left to right along the line of height h_0 . (3) A vertical line is then drawn in the middle to cut the strip into two equal halves (note this line may cut an item that is packed partially in the right half). Draw two horizontal line segments of length one half, one across the left half (called the left baseline) and one across the right half (called the right baseline) as low as possible such that the two lines do not cross any item. (4) Choose the left or right baseline which is of a lower height and pack a level of items into the corresponding half of the strip until the next item is too wide. A new baseline is formed and Step (4) is repeated on the lower baseline until all items are packed.
- *TPIM*: touching perimeter (see [9] and [10]). 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_WO_R*: 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 all these algorithms *BASIC* is the fastest (shortest generation time), *TPIM* 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 between buildings that should go smoothly together.

Up to Google Earth 6.0.1, 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, which creates 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.



Fig. 4: Object exported in the COLLADA display form being highlighted on mouse over.

3.1.2.2 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.

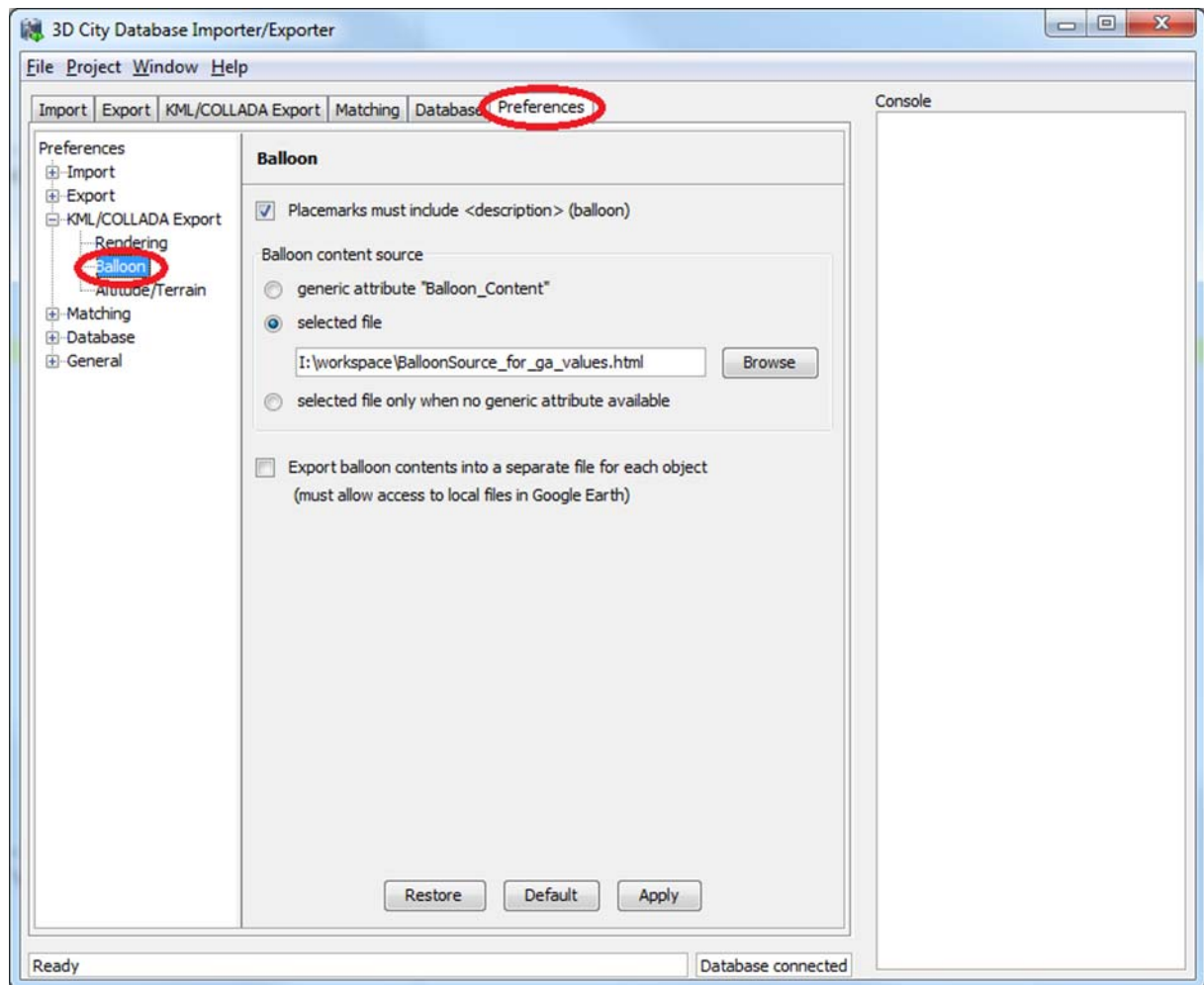


Fig. 5: Balloon settings.

The balloon contents can be dynamically adapted for each city object at export time in a way similar to how Active Server Pages (ASP) [5] work. Placeholders embedded in the HTML template, beginning with `<3DCityDB>` and ending with `</3DCityDB>` tags, will be replaced in the resulting balloon content 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 written 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 [1] 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 aggregate 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.

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> 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/URI  
[ATTRNAME='Illustration']</3DCityDB>" width=400>
```

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

```

```



Fig. 6: 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 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>");
  }

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

</script>
```

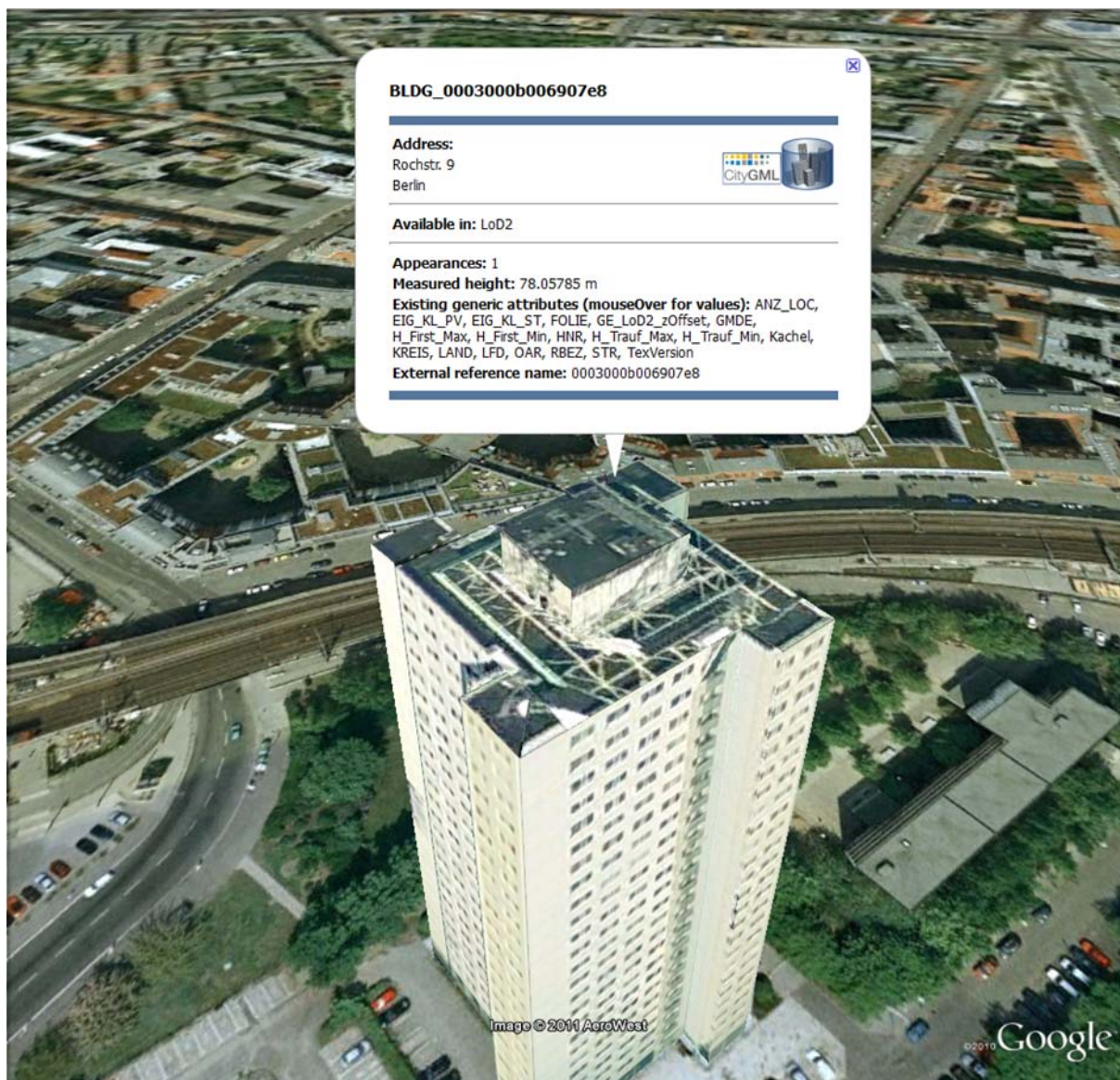



Fig. 7: Model placemark with dynamic balloon contents showing the list of generic attributes.

3.1.2.3 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.

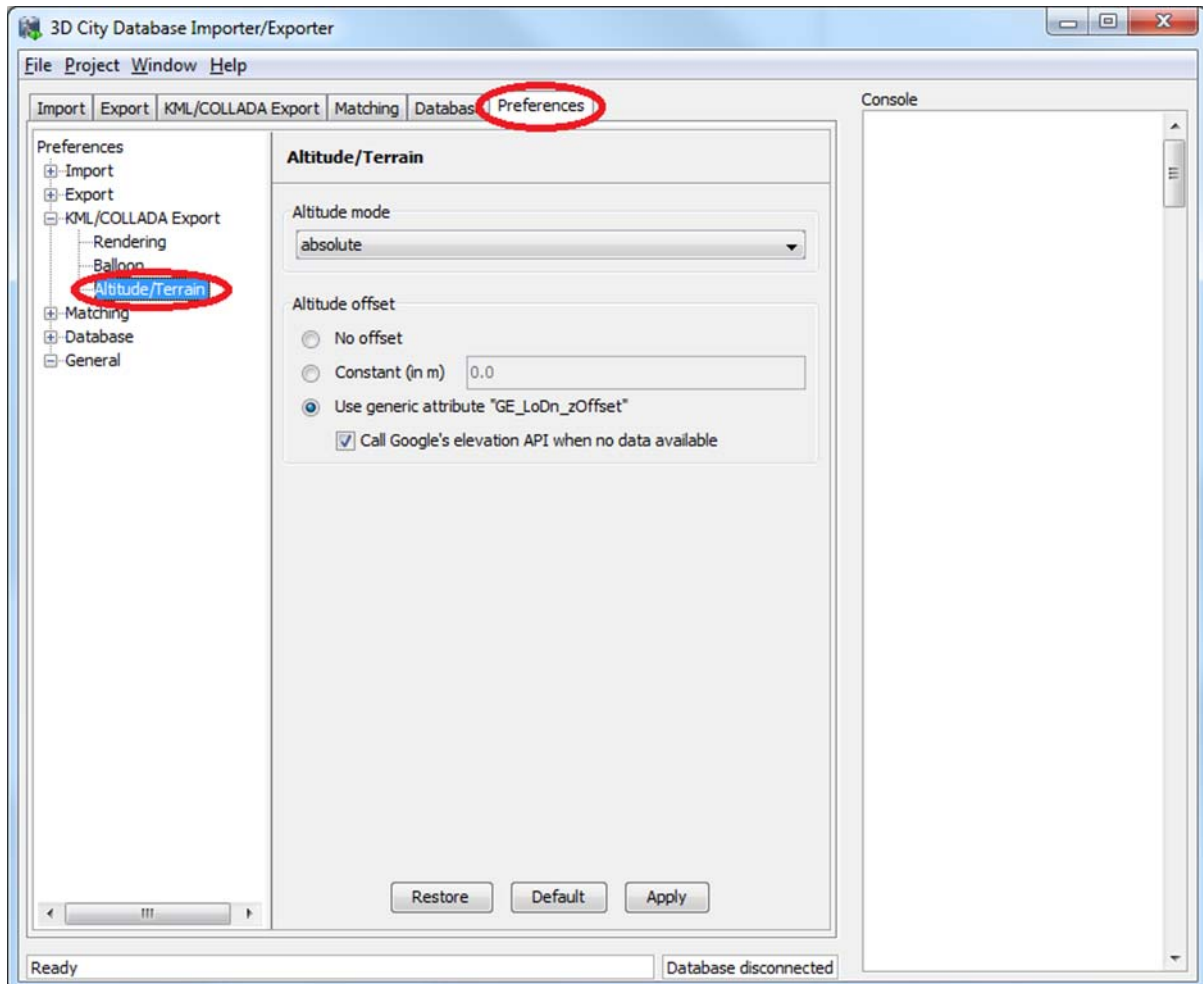


Fig. 8: Altitude/Terrain settings.

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.

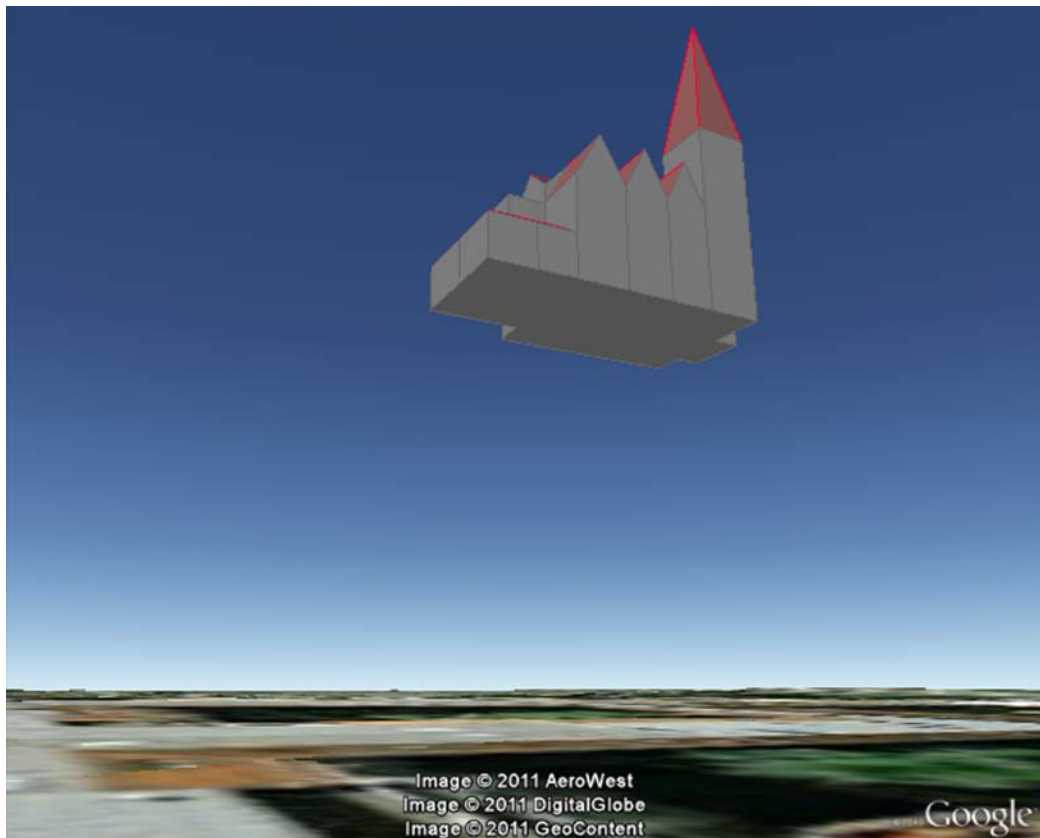


Fig. 9: Possible export result with relative altitude mode.

When *absolute* altitude mode is chosen, the z-coordinates of the exports represent the vertical distance from the vertical datum - the ellipsoid which most closely approximates the Earth curvature, for Google Earth this is the WGS84 EGM96 Geoid, see KML documentation [3] -, 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 7, 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 [4]. 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.

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: wrong height data of the model, 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.

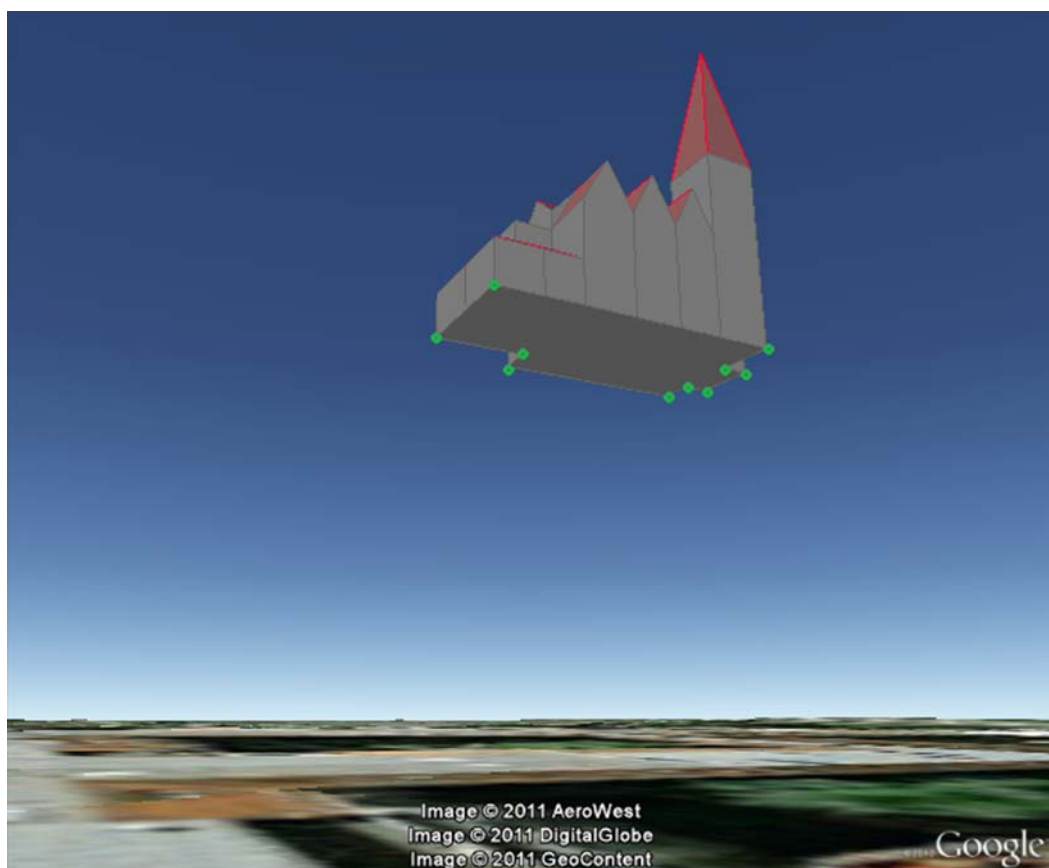


Fig. 10: Points sent to Google's Elevation API for calculation of the zOffset.

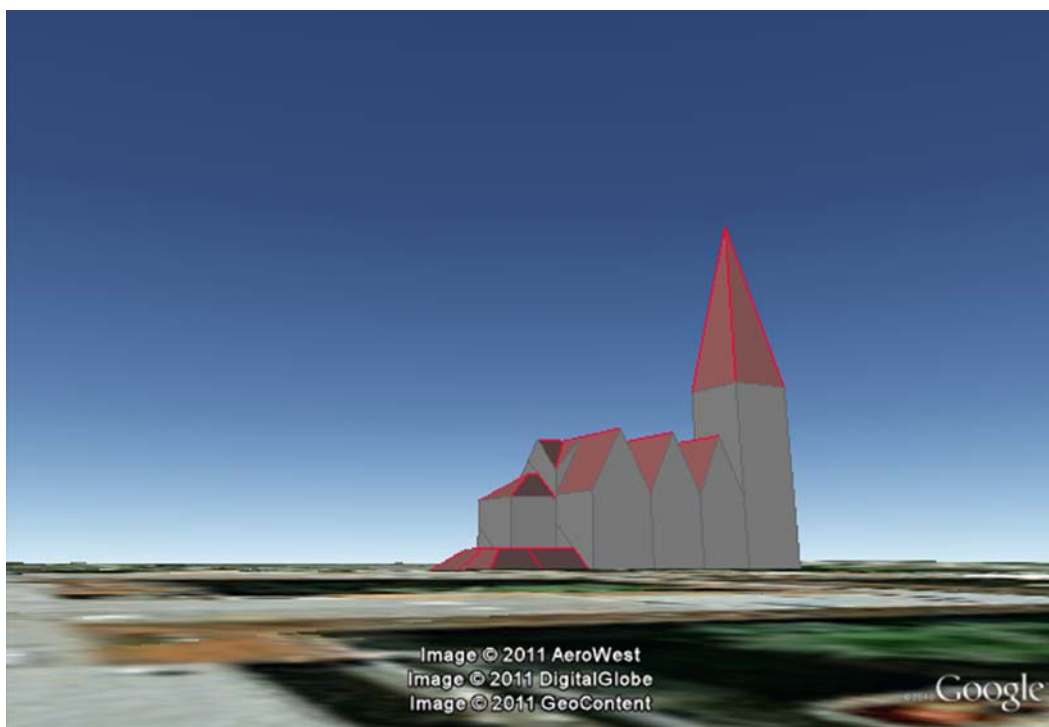


Fig. 11: Export with *absolute* altitude mode and *no offset*.

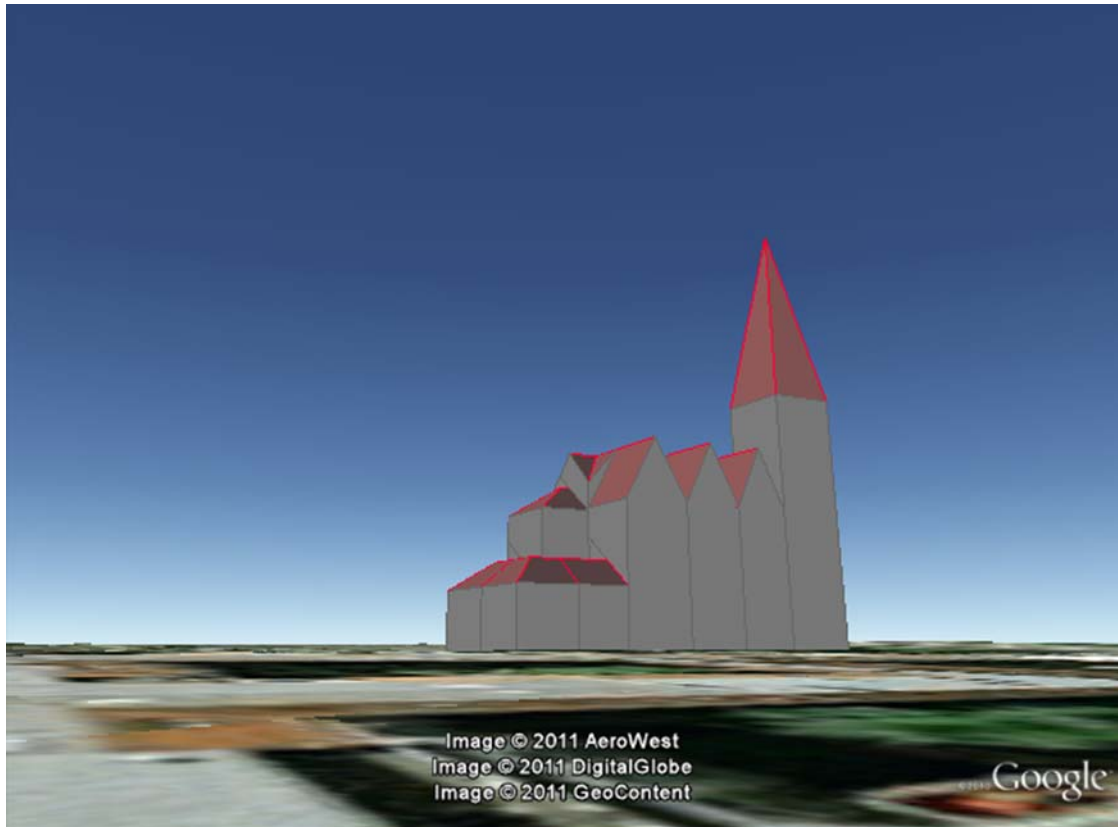


Fig. 12: Export with *absolute* altitude mode and use of *GE_LoDn_zOffset*.

3.1.3 Batch mode

As for the CityGML imports and exports, KML/COLLADA exports can also be executed from the shell without opening a graphical user interface. The command line syntax is:

```
java [options] -jar impexp.jar -shell -kmlExport  
<export_filename> -config <config_filename>
```

[options] refers to all possible settings in the Java Virtual Machine (JVM) itself. One recommended option is `-Xmx1024m`, which sets the maximum heap size to 1GB. The user can adapt this value according to his/her hardware for optimal performance. Bigger heap sizes mean less often garbage collections and therefore improved performance. For a comprehensive list of all JVM options see [7].

- `-shell` tells the Import/Export tool to execute in a shell environment without graphical user interface.
- `<export_filename>` indicates the target file for the resulting exports.
- `<config_filename>` points to a configuration file containing the project settings. Configuration files can be explicitly created and stored by the Importer/Exporter tool by clicking on the menu "*Project*", "*Save Project As...*". Configuration files are written in .xml format and can be adapted with any text editor.

3.1.4 Loading exported models in Google Earth

Usage of the most up-to-date Google Earth version is highly recommended. Some of the features described in this documentation, like highlighting, require version 6.0.1 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.

Recommended graphics mode ("*Tools*", "*Options*", "*3D View*") on Windows platforms is DirectX.

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 3.1.2.3), 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 3.1.2.1, 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.

3.1.5 General setting recommendations

Important note: Although the KML/COLLADA exporter uses (and requires) building objects with 3D coordinates, it does currently not support truly three-dimensional coordinate reference systems in Oracle as input for the KML/COLLADA export, even if the underlying DB supports them (starting from Oracle 11g R1), see chapter 3.2 for a comprehensive explanation.

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. Kml format seems to have a positive effect on the stability of the Earth browser.
- 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 load all data no matter whether it is visible or not.
- 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 8: 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.

3.2 Support for Coordinate Reference Systems (CRS)

3.2.1 General information

When setting up a new instance of the 3D City Database, a Coordinate Reference System (CRS) has to be provided as mandatory setup parameter. This CRS is used as default reference system for all spatial objects which are created and stored in the database instance as well as for building spatial indexes and performing spatial functions.

Oracle provides comprehensive support for different types CRSs. Both Oracle 10g and 11g support 2D coordinate systems which are classified into two types: *Geographic2D* and *Projected*. The first type of coordinate system specifies the longitude and latitude on the earth surface approximated by a reference ellipsoid and is also referred to as *Geodetic* coordinate system. *Projected* types of 2D coordinate systems specify how to project longitude and latitude values on a reference *Geographic2D* system to a two-dimensional Euclidean coordinate system. Starting from Oracle 11g, support for 3D coordinate systems has been added which are classified into the following three types: *Geographic3D* (*Geographic2D* plus ellipsoidal height), *Geocentric* (specifies x,y,z values with reference to the center of the earth), and *Compound* (combines either *Geographic2D* or *Projected* (2D) with a vertical coordinate system specifying height based on gravity, above mean sea level, etc.).

The 3D City Database and the Importer/Exporter brought by this release are designed to support both Oracle 10g (R2) and Oracle 11g (R1 and R2). Although it is possible to specify a 3D coordinate system when setting up a new database instance, it is **strongly recommended** to pick a 2D coordinate system (*Geographic2D* or *Projected*) instead. The reason for this is that many features of the Importer/Exporter have been implemented to be backwards compatible with Oracle 10g R2, and thus require a 2D coordinate system to work properly. For example, the newly introduced KML/COLLADA export capability performs a coordinate transformation to WGS84 using a 2D target coordinate system (following the EPSG 4326 definition). This target 2D coordinate system cannot be changed for backwards compatibility reasons because it is available in both Oracle 10g and 11g. If, however, the 3D City Database is set up on Oracle 11g with a 3D coordinate system, the coordinate transformation will yield an error message because the dimensionality of the source coordinate system differs from the dimensionality of the target coordinate system which is not allowed on Oracle 11g.

Regardless of the CRS associated with a spatial object, the spatial data types offered by both Oracle 10g and 11g fully support three-dimensional coordinate values. This is important with respect to the storage of geometries in the 3D City Database: Since the 3D City Database is meant to store and manage virtual 3D city models based on CityGML, and CityGML is a true 3D standard, **all geometries are stored with three-dimensional coordinate values** in the 3D City Database. There are only very few exceptions to this rule, where geometries with two-dimensional coordinate values are allowed in CityGML (please refer to the CityGML specification [2] for details).

Note: Upcoming versions of the Importer/Exporter will bring full support for 3D coordinate systems on Oracle 11g.

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

The definition of the CRS for setting up a new instance of the 3D City Database consists of two components: 1) a valid Oracle *Spatial Reference Identifier* (SRID) and 2) an OGC GML conformant *definition identifier for the CRS*. Both components are stored in the table DATABASE_SRS. The CRS definition is fixed and shall not be changed at any later point in time (please refer to the chapter 3.2 of the previous 3D City Database 2.0.1 documentation [1] for more information about the setup procedure).

The SRID is an integer value key pointing to spatial reference information within Oracle's MDSYS.CS_SRS table. Oracle is 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 MDSYS.CS_SRS. As discussed in the previous section, please choose a *Geographic2D* or *Projected* CRS.

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* [13]. 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 [13]. 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 SRID and CRS URN encoding 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).

3.2.3 Management of user-defined CRSs

The previous version 1.2.2 of the Importer/Exporter already introduced the possibility to pass coordinate values given in a different CRS than the internal database CRS as input values for a spatial bounding box filter when importing/exporting CityGML documents. However, the support for further CRSs was still preliminary in such that a user could only choose from a predefined and fixed number of CRSs.

This release of the Importer/Exporter brings full support for user-defined CRSs. Similar to the CRS information that has to be provided at setup time of a new 3D City Database instance, a user-defined CRS consists of both an Oracle SRID and a GML conformant URN encoding of the CRS. For the management of user-defined CRSs, a new user dialog has been added to the *Preferences* tab as shown in the Fig. 13.

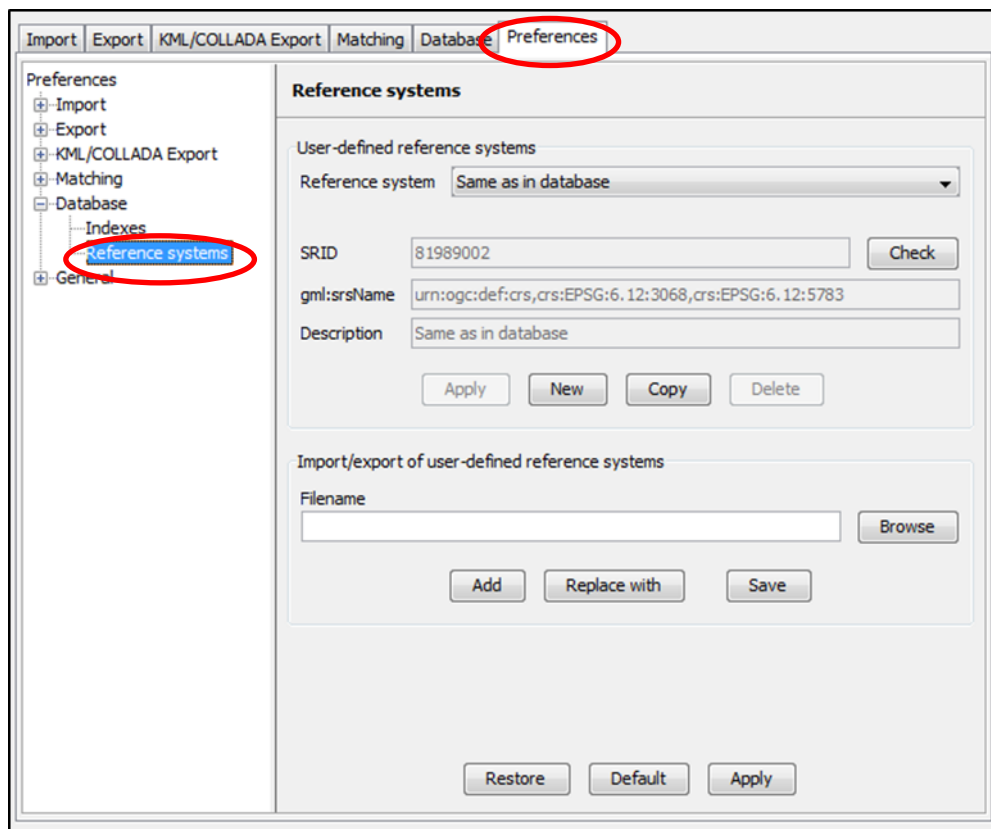


Fig. 13: Support for user-defined CRSs in the *Preferences* tab.

The management of user-defined CRSs can be found in the *Reference systems* subnode of the *Database* preferences node (the latter one is also introduced with this release, cf. chapter 3.5). On top of the preferences page, a combo allows for choosing a CRS for display and editing from the list of user-defined CRSs. The 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, the fields of the *Same as in database* entry cannot be edited (cf. Fig. 13.).

A new user-defined CRS can be added to this list after clicking the *New* button. Please provide an Oracle SRID in the corresponding *SRID* input field of the user dialog and pass a corresponding URN encoding to the *gml:srsName* input field (optional). A short, meaningful textual description of the CRS has to be given in the *Description* field. This description is used as key value for the list of user-defined CRSs displayed in the combo box on top (and also in similar combo boxes on further tabs of the Importer/Exporter). The new CRS is added to the list of user-defined CRSs by clicking on the *Apply* button. The following screenshot provides an example.

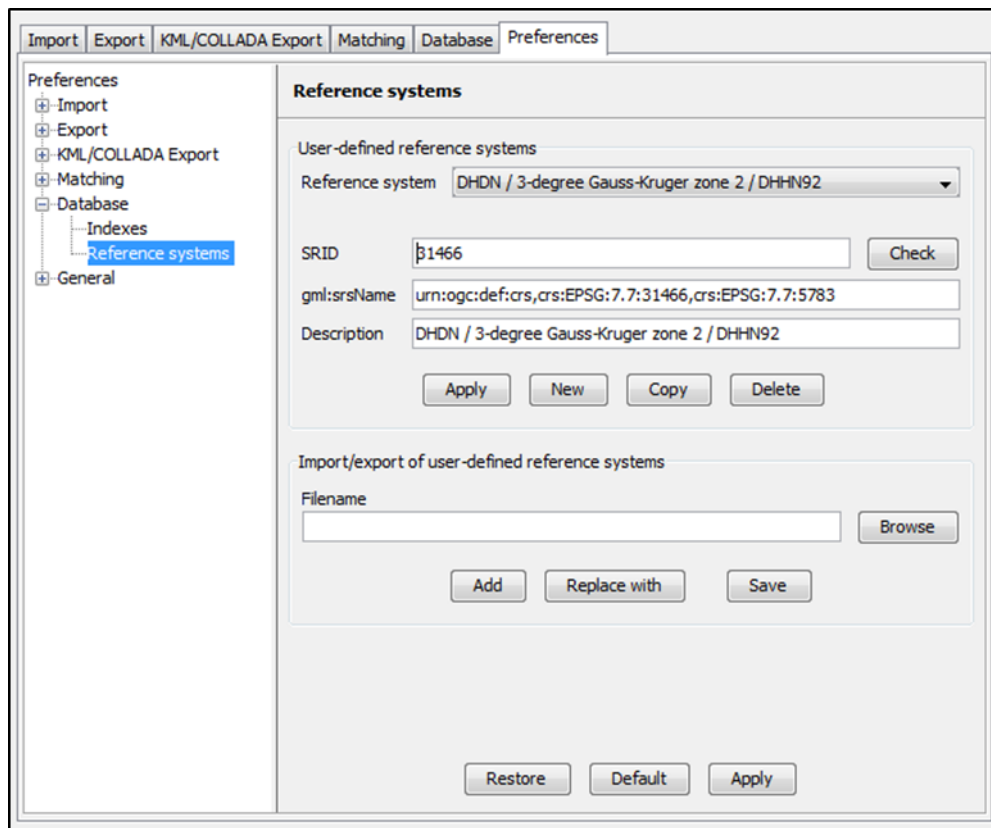


Fig. 14: Adding a new CRS to the list of user-defined CRSs.

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. 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 due to the fact that a) the list of predefined spatial reference systems differs between different versions of Oracle Spatial (e.g., between 10g R2 and 11g R2) and b) the fact, that Oracle also supports the definition of user-defined spatial reference systems (please check the Oracle Spatial documentation for further guidance on how to create spatial reference systems in Oracle). In order to add a user-defined CRS to the Importer/Exporter which is not supported by the underlying Oracle Spatial database by default, this CRS has first to be registered with the internal Oracle tables. As soon as the CRS is announced to Oracle, 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 project settings of the Importer/Exporter and loaded upon application start. It can additionally be exported into an extra file without the remaining project settings. This allows for easily sharing user-defined CRSs between different Importer/Exporter installations. Please provide a valid filename in the corresponding input field *Filename* (clicking on the *Browse* button opens a dialog which allows for choosing a file) and click on *Save*. There are two 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 of `templates/CoordinateReferenceSystems` within the installation folder of the Importer/Exporter. Each CRS definition is stored in its own file and, thus, can be easily imported and added to the list of user-defined CRSs. The URN encoding of the predefined CRSs generally lacks a height reference system which has to be added before using this CRS as target reference system for CityGML exports (cf. chapter 3.3.1 for more details).

3.2.4 Usage of user-defined CRSs

User-defined CRSs can be used at the following locations within the Importer/Exporter:

1. Defining the CRS for the bounding box filter on the CityGML *Import* tab,
2. Defining the CRS for the bounding box filter on the CityGML *Export* tab,
3. Defining the CRS for the bounding box filter on the *KML/COLLADA Export* tab (cf. chapter 3.1), and
4. Defining the target CRS for a coordinate transformation during CityGML exports on the *Export* tab (cf. chapter 3.3.1).

The following Fig. 15 shows an example usage of the CRS which has been added to the list of user-defined CRSs in the previous chapter. The CRS is applied to the spatial bounding box filter on the CityGML *Import* tab by choosing the corresponding value from the combo box. The Importer/Exporter will use this CRS information for the interpretation of the coordinate values passed to the *Xmin*, *Xmax*, *Ymin* and *Ymax* input fields and automatically transform them to the internal CRS of the 3D City Database instance in order to apply the spatial filter.

Often the bounding box used as a spatial filter for a CityGML import/export or KML/COLLADA export is derived from querying a public web-based map service such as Google Maps, Microsoft Bing Maps, or OpenStreetMap. Usually, points on these maps are latitude and longitude values according to the WGS84 ellipsoid and datum. If you want to directly make use of the map coordinate values in the spatial bounding box filters of the Importer/Exporter, you first have to add a user-defined CRS for WGS84 and choose this as reference system for the corresponding filter. The Importer/Exporter is shipped with a predefined CRS definition for WGS84. This can be loaded from an external file which is located in the subfolder `templates/CoordinateReferenceSystems` within the installation folder of the Importer/Exporter.

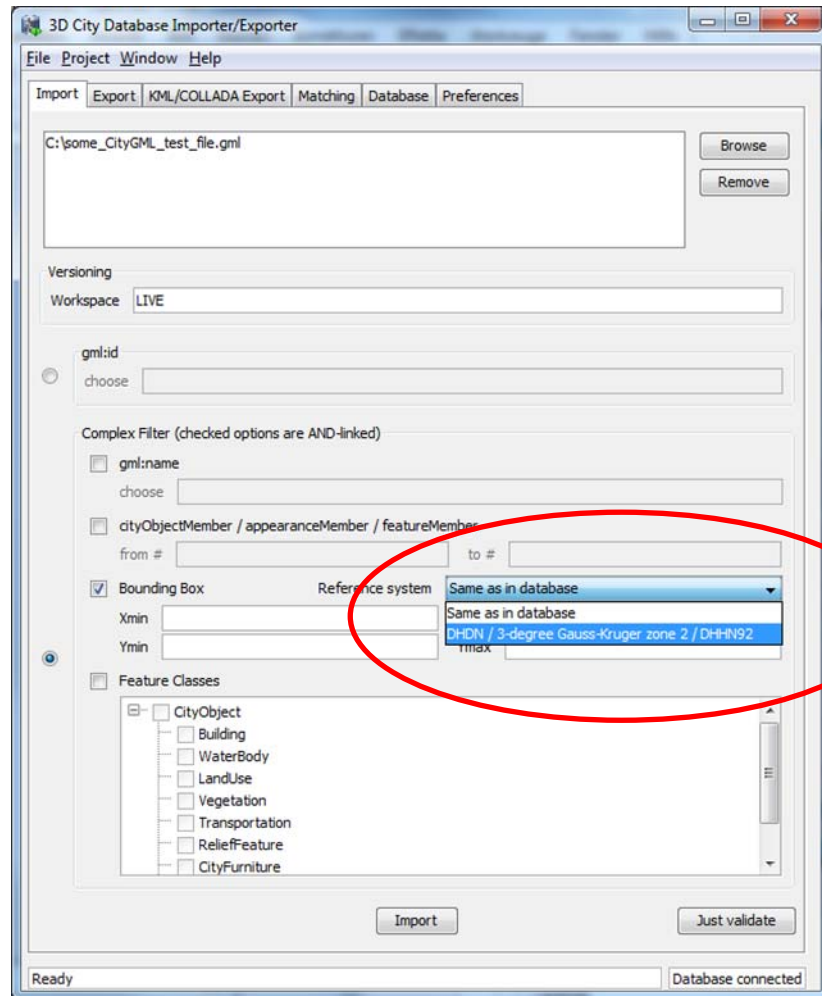


Fig. 15: Example usage of a user-defined CRS for specifying the reference system of the spatial bounding box filter applied to a CityGML import.

Since not every user-defined CRS is necessarily supported by the Oracle Spatial database, the Importer/Exporter verifies all CRSs upon every database connection. If a user-defined CRS is not supported by the Oracle Spatial database underlying the 3D City Database, a corresponding warning message is logged in the console window. The CRS combo boxes on the CityGML *Import/Export* tab and *KML/COLLADA Export* tab are smart in such they only offer a subset of CRSs which is supported by the currently connected database.

3.3 CityGML Export Enhancements

With this release of the Importer/Exporter, two new features have been added to the CityGML export functionality: 1) performing a coordinate transformation to an arbitrary user-defined CRS, and 2) exporting the database content into tiles.

3.3.1 Coordinate Transformation

As discussed in the previous chapter 3.2, the geometry of city objects stored in the 3D City Database is internally associated with a fixed Coordinate Reference System (CRS) that has to be defined at database setup and shall not be changed afterwards.

Previous versions of the Importer/Exporter could only export data in that same CRS. The 1.3.0 release of the Importer/Exporter now allows applying a coordinate transformation into another CRS during CityGML exports. Please provide a user-defined CRS as target reference system (cf. chapter 3.2 for guidance on the management of user-defined CRSs).

A new CRS combo box has been added at the top of the CityGML *Export* tab. Simply choose the target reference system from the list of CRSs as shown in Fig. 16.

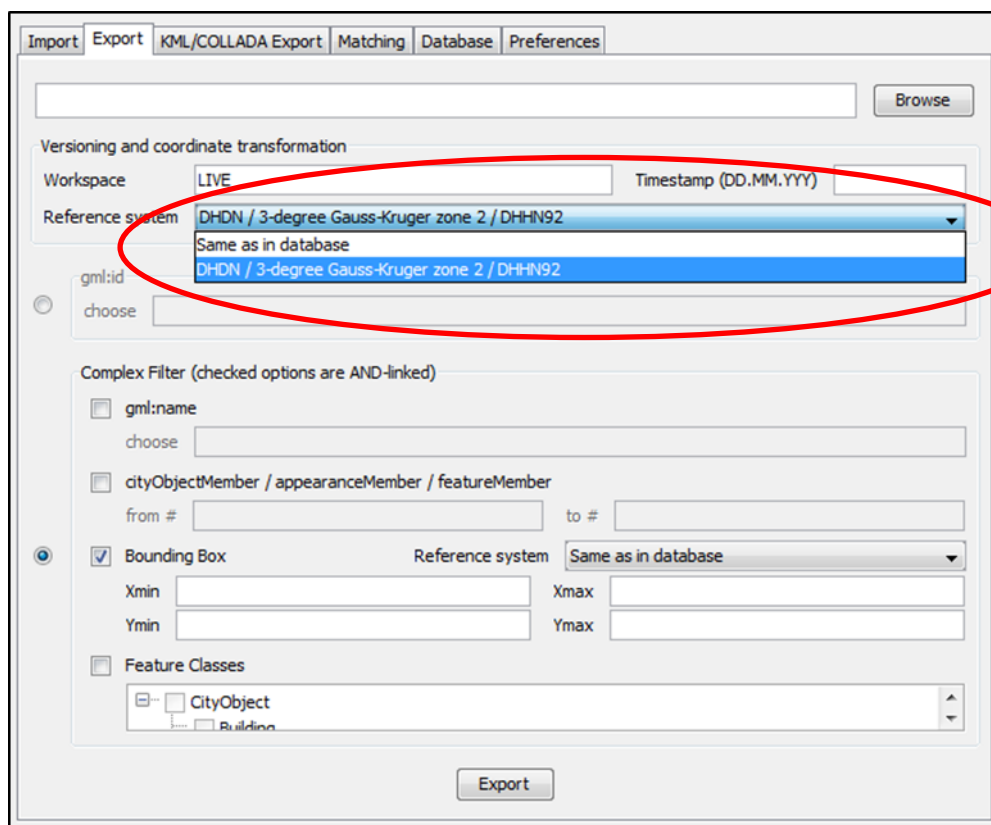


Fig. 16: Choosing a user-defined CRS as target reference system for a coordinate transformation applied during the CityGML export.

Note: For CityGML imports there is currently no automatic or user-definable coordinate transformation into the internal CRS of the 3D City Database instance available. Please make sure that the spatial reference system specified in the CityGML instance document matches the CRS of the 3D City Database instance *prior to importing*

data. If the CRS of the CityGML instance document to be imported does not match, the following workaround procedure can be applied:

- 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 during the export as explained above.
- 4) The exported CityGML document now matches the CRS of the target 3D City Database instance and can be imported into that database.

Support for coordinate transformations during CityGML imports will be added in future versions of the Importer/Exporter.

3.3.2 Tiling

When exporting the entire 3D city model stored in the 3D City Database into 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 large to open in 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.

Starting from its first release, the Importer/Exporter allows for applying a spatial bounding box filter to CityGML exports which helps in reducing the number of exported features and thus of the resulting file size. If however the area of interest is still comparatively large the user had to manually divide the area into smaller areas.

This release of the Importer/Exporter provides the possibility to automatically tile the area specified by a bounding box filter. A corresponding user dialog has been added to the *Preferences* tab. It is available as subnode of the *Export* preferences node as shown in the following Fig. 17.

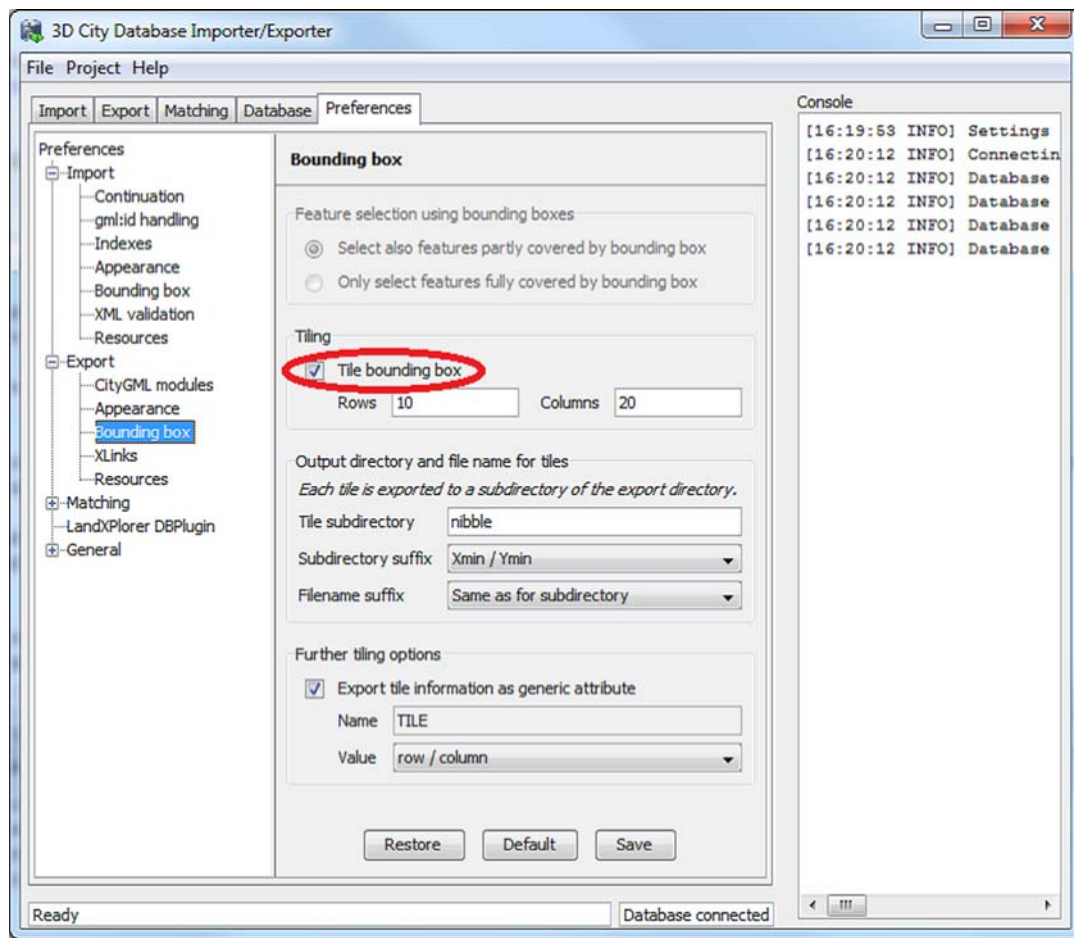


Fig. 17: Tiled export options under the *Preferences* → *Export* tab.

Tiled exports are only available with the bounding box filter being enabled on the CityGML *Export* tab (further filters can of course additionally be used). To make use of the new tiling feature, please check the *Tile bounding box* option. If this option is unchecked, the only available settings for the bounding box filter are shown in the section *Feature selection using bounding boxes* on top of the user dialog. These settings have already been introduced in the first release of the Importer/Exporter and are used to indicate whether only objects fully covered by the bounding box will be exported or also those partially covered by it.

When tiling is selected, the *Feature selection using bounding boxes* section becomes inactive and all other options of this preferences tab are available. First, 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 (column ENVELOPE of the table CITYOBJECT) has to be either inside or on the left or top border of the tile.

When exporting, a subdirectory will be created for each tile. These subdirectories are located beneath the main export directory specified on the CityGML *Export* tab. They all share a common free choosable name part and a tile specific suffix. 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 3.3.1, 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. Thus it will be easier to track which object belongs to which tile, since the coordinates of the objects contained in the tile are exported in the same reference system. The file name of the CityGML instance document created per tile may also receive a tile specific suffix. If the corresponding option is selected, the suffix will be the same as for the tile directory.

For further traceability it is possible to attach a CityGML generic attribute called *TILE* to each exported 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 new tiling feature enabled, a bounding box spanning the overall area of the model has to be provided. This bounding box can be easily calculated using a new database operation introduced in this release which is available on the *Database* tab (cf. chapter 3.5).

Note: Using the center of an object's envelope as topographical criterion for a tiled export leads to a side-effect when tiling is combined with the feature count filter on the CityGML *Export* tab: the number of objects being exported can no longer be exactly determined since the calculation of the object's envelope center must be done at a later point in time. Therefore, the feature count filter only sets a possible maximum value in this filter combination. For instance: when set to export cityObjectMember / appearanceMember / featureMember from #1 to #500, only #493 may be reached without the 3D City Database Importer/Exporter reporting any errors, since the missing 7 objects were discarded after being queried due to their envelope center not being within the tile.

3.4 Rework and Redesign of the Matching/Merging Tool

The Matching/Merging tool has been substantially reworked in this release of the 3D City Database and the Importer/Exporter. First, identified bugs have been fixed in the underlying PL/SQL database procedures. Second, the user interface has been redesigned in order to improve usability. The goals of the Matching/Merging tool as well as its mode of operation and process workflow have not been changed in this release. Thus, the general description of the tool given in chapter 5 of the previous 3D City Database 2.0.1 documentation [1] still holds for this release. This section only presents additions to the former documentation.

User interface changes

The following Fig. 18 sketches the changes on the main tab *Matching* of the Importer/Exporter. Most of the changes result from rearranging GUI elements without modifying their original meaning. Settings related to candidate and master buildings have been grouped together to better grasp the input and output parameters for the matching/merging process.

The left hand side of Fig. 18 shows the previous design of the Matching/Merging tool as contained in version 1.2.2 of the Importer/Exporter. The redesigned layout shipped with version 1.3.0 is shown on the right. User input fields have been numbered. Input fields with identical meaning share the same number (shown in red). Green numbers are used to mark new input fields which have been introduced with this release.

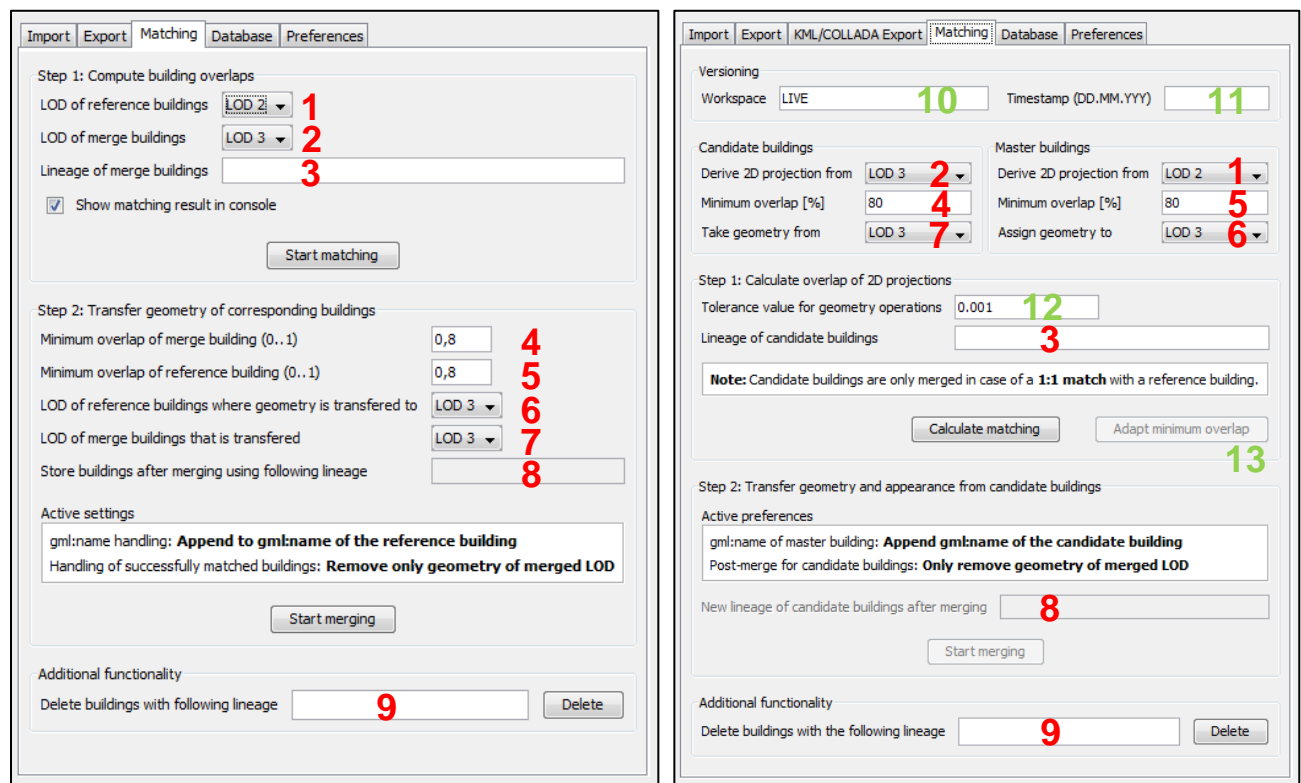


Fig. 18: Redesigned user interface of the matching/merging tool (left: the previous user interface as contained in version 1.2.2 of the Import/Export tool, right: the current user interface as contained in version 1.3.0 of the Import/Export tool). Corresponding input fields share the same numbering.

The Matching/Merging tool has been augmented with the following input fields and functionalities:

- **Support for version-enabled databases**

Similar to other tabs of the Import/Export tool, the input group *Versioning* has been added on top of the *Matching* tab. The two new fields *Workspace* (10) and *Timestamp* (11) allow for specifying an Oracle workspace on which the matching/merging process shall be executed (default is *LIVE*).

- **Tolerance value for geometry operations**

The tolerance input field (12) allows for defining a tolerance value which is passed to Oracle's spatial functions used in the matching/merging process. Tolerance is used to associate a level of precision with spatial data and reflects the *distance that two points can be apart and still be considered the same* (for example, to accommodate rounding errors; please check the *Oracle Spatial User's Guide and Reference* for further information). The tolerance value is a number of the units that are associated with the internal CRS of the 3D City Database. For example, if the unit of measurements is meter, a tolerance value of 0.005 indicates a tolerance of 0.005 meter (that is, 1/200 meter), and a tolerance value of 2 indicates a tolerance of 2 meters. Please make sure to pick a reasonable value that fits your data.

- **Adapt minimum overlap**

Whether a candidate building and a master building are considered a match depends on the user-defined minimum overlap values for their projected 2D geometries (fields 4 and 5; cf. chapter 5.3 of the 3D City Database 2.0.1 documentation [1]). In the former 1.2.2 release, changing these values for a given set of candidates and masters required the restart of the entire matching procedure. The new button “*Adapt minimum overlap*” (13) allows for quickly influencing the matching result based on new minimum overlap values.

There are also minor changes to the *Preferences* section of the matching/merging tool as shown in Fig. 19.

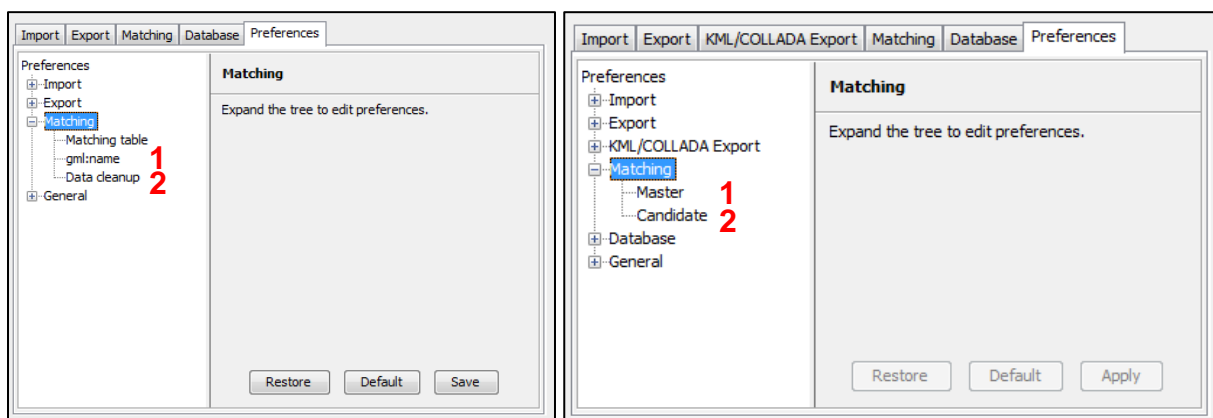


Fig. 19: Reworked preferences section of the matching/merging tool (left: the previous user interface as contained in version 1.2.2 of the Import/Export tool, right: the current user interface as contained in version 1.3.0 of the Import/Export tool). Corresponding input fields share the same numbering.

The preferences node “*gml:name*” has been renamed to “*Master*” since its settings only affect the master buildings in the merging process. For the same reason, “*Data cleanup*” has been renamed to “*Candidate*”. The contents of both preferences nodes remain the same as in the previous 1.2.2 release. Finally, “*Matching table*” has been entirely removed from the preferences section.

Bug fixes

The underlying PL/SQL procedures of the matching/merging tool have been completely reworked for this release to fix identified bugs:

- **Support for both Oracle 11g and 10g**

The former version of the Matching/Merging tool was only designed for Oracle 10g R2. The matching/merging process will abort with errors on Oracle 11g R1 and Oracle 11g R2 when 3D coordinate reference systems are used for spatial objects. With this release, these bugs have been fixed and the PL/SQL procedures now bring full support for Oracle 11g.

- **Support for version-enabled databases**

The previous version of the Matching/Merging tool does not correctly work on databases with version-enabled tables. This bug has been fixed with this release.

- **Issues with moving appearances from candidates to master buildings**

In some rare situations, appearances were not correctly moved from the candidate building to the matched master building. The processing of appearances has been reworked to fix this bug.

The new PL/SQL procedures of the matching/merging tool are contained in the 2.0.5 distribution package of the 3D City Database (check files `MATCH.sql` and `MERGE.sql` in the subfolder `./oracle_spatial/PL_SQL/GEODB_PKG/MATCHING`).

3.5 Extensions to the Database tab and preferences

The *Database* tab of the Importer/Exporter has been augmented with a *Database operations* section marked in red in the following Fig. 20.

The screenshot shows the 'Database' tab of the Importer/Exporter interface. At the top, there are tabs for 'Import', 'Export', 'KML/Collada Export', 'Matching', 'Database', and 'Preferences'. The 'Database' tab is selected and highlighted with a red circle. Below the tabs, there is a 'Connection' dropdown menu set to 'New connection'. Under 'Connection details', there are input fields for 'Description' (containing 'New connection'), 'User name', 'Password', and 'Server'. There are also buttons for 'New', 'Copy', 'Delete', and a checkbox for 'Save Password'. The 'Port' field is set to '1521' and the 'Database' field is empty. At the bottom of this section are 'Disconnect' and 'Info' buttons. Below this is the 'Database operations' section, which is highlighted with a red rectangle. It contains a 'Workspace' dropdown set to 'LIVE', a 'Timestamp (DD.MM.YYY)' field, and two radio buttons: 'Generate database report' and 'Calculate maximum bounding box' (which is selected). Below these are two dropdown menus: 'Top-level feature' set to 'CityObject' and 'Reference system' set to 'Same as in database'. An 'Execute' button is at the bottom of this section.

Fig. 20: Additions to the *Database* tab of the Importer/Exporter.

The following operations can be executed from this section:

- **Generate database report**

This functionality is already included in the previous 1.2.2 release of the Importer/Exporter and has not been changed with this release. The output is a list of all tables and the number of contained data rows printed to the console window.

- **Calculate maximum bounding box**

This operation allows for the calculation of the maximum bounding box of all city objects within the database. The resulting 2D coordinates of the bounding box (given as lower left corner and upper right corner) are printed to the console window and can be used, for example, as input for the bounding box filters offered by the CityGML import and export as well as the KML/COLLADA export (you will need to copy&paste the values).

The input city objects for the calculation of the bounding box can be restricted to a specific CityGML top-level feature type (e.g., Building, WaterBody, etc.). Furthermore, the resulting bounding box coordinates may be transformed to a different

coordinate reference system (cf. chapter 3.2 of the release notes for a description of the new support for user-defined CRS introduced with this release).

The user can define an Oracle workspace on which both operations shall be executed (default is *LIVE*). Similar to other tabs of the Importer/Exporter, the two input fields *Workspace* and *Timestamp* are used to specify the target Oracle workspace.

The following Fig. 21 shows an example for the calculation of a maximum bounding box of a 3D City Database instance. The resulting bounding box is additionally transformed to the WGS84 reference system. For this purpose, a corresponding user-defined CRS (WGS84) was created.

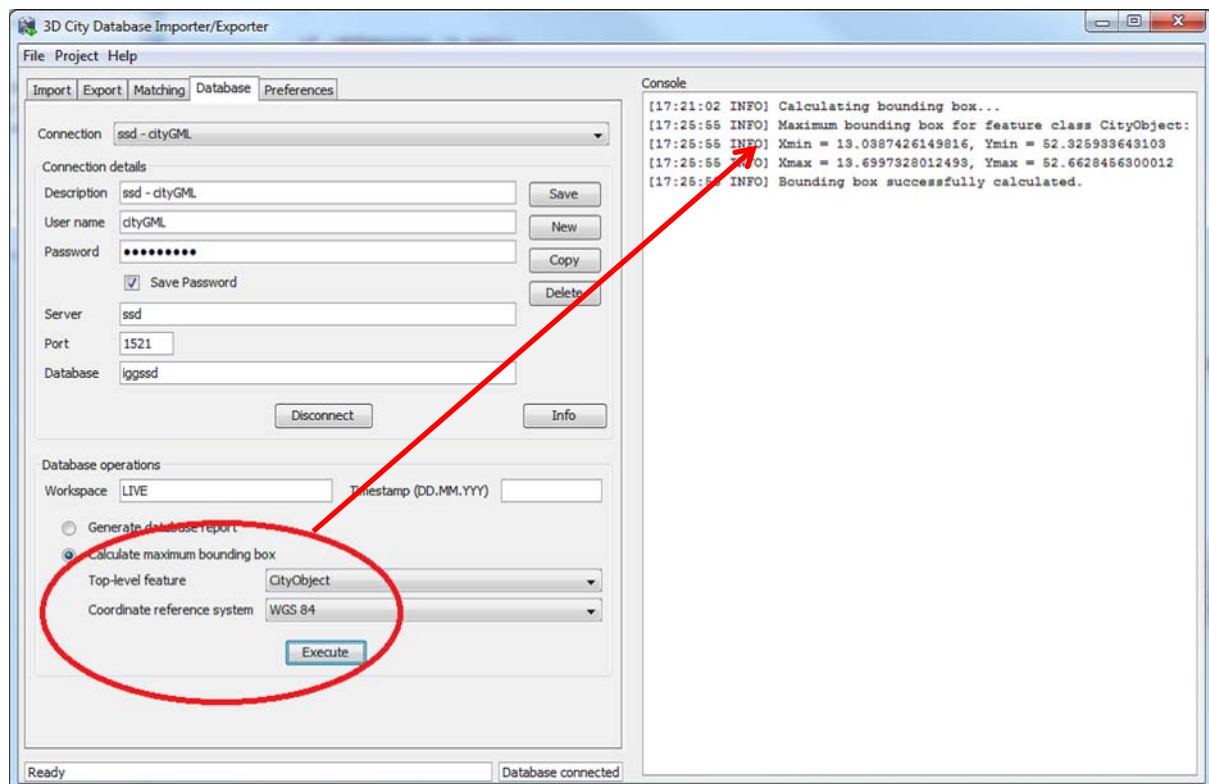


Fig. 21: Calculate maximum bounding box of a database workspace under the *Database* tab.

In addition, a new *Database* preferences node has been introduced on the *Preferences* tab of the Importer/Exporter as shown in the Fig. 22. The *Indexes* subnode has been copied 1:1 from the *Import* node to the new *Database* node. Please refer to chapter 4.1.2.4 of the previous 3D City Database 2.0.1 documentation [1] for a description of its settings and functionality.

The *Reference systems* subnode facilitates the support and management of user-defined coordinate reference systems. Please refer to chapter 3.2 of these release notes for more information.

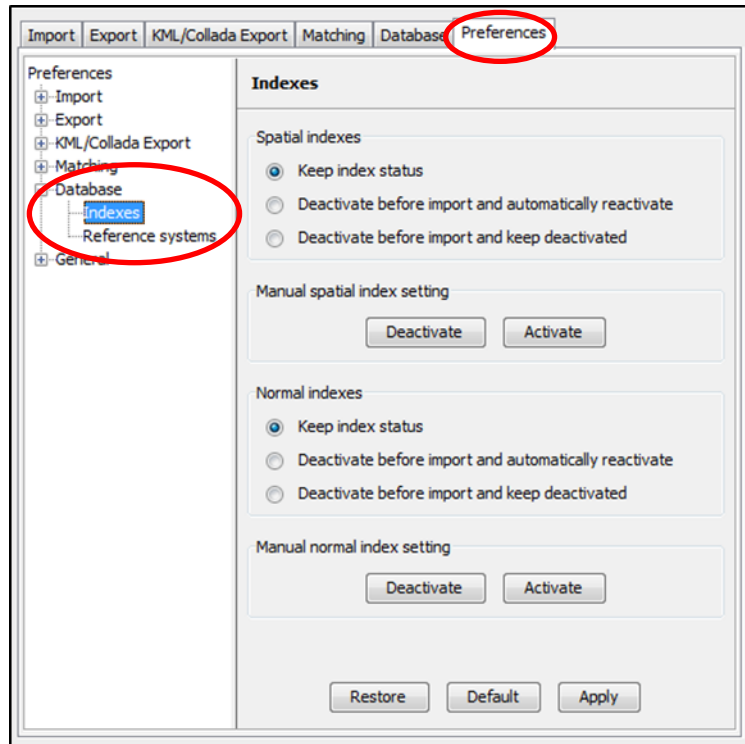


Fig. 22: New *Database* preferences node on the *Preferences* tab of the Importer/Exporter.

3.6 New PL/SQL functionality

3.6.1 PL/SQL package GEODB_DELETE

Starting from its first release, the 3D City Database has been shipped with a set of PL/SQL packages which are automatically installed during the setup procedure of the 3D City Database. These packages are required by the Importer/Exporter. The latest version of these PL/SQL packages shipped with the version 2.0.5 of the 3D City Database is a *mandatory dependency* of the Importer/Exporter version 1.3.0.

Moreover, the PL/SQL packages and their provided functionality can also be used to build third-party applications on top of the 3D City Database. The following table enumerates the available PL/SQL packages together with a short description. All package names share the common prefix GEODB.

Package name	Description
GEODB_IDX	Provides functions to create, drop, and check both spatial and non-spatial indexes on tables of the 3D City Database.
GEODB_MATCH and GEODB_MERGE	GEODB_MATCH implements the required functions and procedures to facilitate the matching of candidate and master buildings in the database. Based on the matching results, the GEODB_MERGE package allows for merging identified buildings. Please check chapter 5 of the previous 3D City Database 2.0.1 documentation [1] as well as chapter 3.4 of this document for more information.
GEODB_STAT	Creates a report on the number of entities currently stored in all tables of the 3D City Database.
GEODB_UTIL	Contains utility functions.
GEODB_DELETE_BY_LINEAGE	Deletes all buildings sharing a common LINEAGE attribute (see table CITYOBJECT). Associated features and geometries will also be deleted.

With the 2.0.5 release of the 3D City Database, a new PL/SQL package called GEODB_DELETE has been added to this list. It provides procedures which facilitate the deletion of single entities within the database.

The procedures take care of the many associations and dependencies between database relations and entities. For example, when deleting an entry from the BUILDING table, also its associated boundary surfaces stored in THEMATIC_SURFACE as well as its addresses stored in ADDRESS are automatically removed. And, of course, also its geometry representations in all LoDs are deleted from SURFACE_GEOMETRY which again triggers the deletion of

appearance information attached to these geometries. Thus, the entire building will be cleanly removed from the database.

This initial release of the GEODB_DELETE package focuses on the deletion of buildings from the 3D City Database. Thus, it currently does not provide procedures to delete other city objects like transportation features, water bodies, city furniture, etc. Future releases will add corresponding delete functions to this package. Please check the following list of all available procedures:

- `delete_surface_geometry(pid number, clean_apps int := 0)`
- `delete_implicit_geometry(pid number)`
- `delete_external_reference(pid number)`
- `delete_citymodel(pid number)`
- `delete_appearance(pid number)`
- `delete_surface_data(pid number)`
- `delete_cityobjectgroup(pid number)`
(Note: this will only delete the CityObjectGroup entry itself but not its members)
- `delete_thematic_surface(pid number)`
- `delete_opening(pid number)`
- `delete_address(pid number)`
- `delete_building_installation(pid number)`
- `delete_room(pid number)`
- `delete_building_furniture(pid number)`
- `delete_building(pid number)`
- `cleanup_appearances(only_global int :=1)`

Most of the procedures take the primary key id value (PID) of the entry to be deleted as input parameter. For `delete_surface_geometry` an optional second parameter `CLEAN_APPS` can be provided. `CLEAN_APPS` acts as a flag taking two values 0 (false) and 1 (true) in order to indicate whether appearance information associated with the geometry to be deleted should also be removed (default is false). Finally, `cleanup_appearances` removes all appearance information from the database which has no associated geometry element. It also takes a flag 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. Please refer to the CityGML 1.0 specification document [2] for a comprehensive description of global and local appearance information in CityGML.

The GEODB_DELETE package is meant as low-level API for the deletion of single objects. More complex delete operations (such as “Delete only a specific LoD from a given building”, or “Delete all buildings within a given address range”) can be built on top of these low-level operations. The GEODB_DELETE_BY_LINEAGE package exemplifies this for the deletion of buildings sharing a common value in the LINEAGE column of the CITYOBJECT table.

In future releases, the GEODB_DELETE package will be accompanied by further low-level APIs to create and modify single objects. This will facilitate to build complex applications based on a straightforward API to the 3D City Database.

3.6.2 Creating Read-Only Users

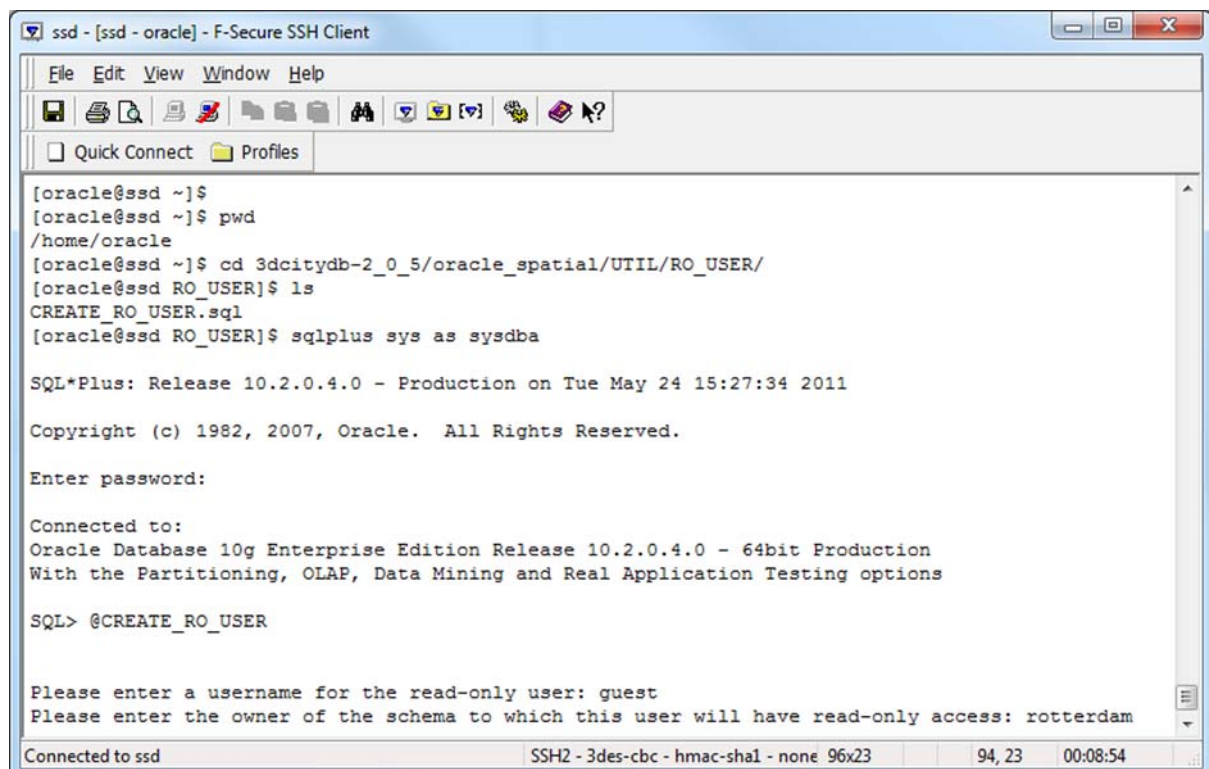
In order to protect your data from unauthorized or accidental modification it can be convenient to define users having read-only access to the 3DCityDB. These users will be allowed to connect to the database and export data in both CityGML and KML/COLLADA formats, but they will be able to neither import new data into the 3DCityDB nor alter the data already stored in the tables in any way.

A dedicated script for the purpose of creating a user with read-only access to the 3DCityDB is included within the 3D City Database 2.0.5 SQL package in case the installation of this package was chosen during the setup process.

Step by step guide through the create read-only user procedure

1. Open a console window.
2. Change the working directory to the folder where the contents of the 3D City Database distribution package are located. In that folder, change to the UTIL/RO_USER subfolder. There is a single file in this subfolder called **CREATE_RO_USER.sql**.
3. Open a SQLPlus connection to the database and login as admin. Any SQLPlus client software may be used.
4. Execute the script `CREATE_RO_USER.sql` by typing `"@CREATE_RO_USER;"` at the SQLPlus prompt.

Note: Make sure to change to the UTIL/RO_USER folder *before* running SQLPlus and executing this command.



```
ssd - [ssd - oracle] - F-Secure SSH Client
File Edit View Window Help
[oracle@ssd ~]$
[oracle@ssd ~]$ pwd
/home/oracle
[oracle@ssd ~]$ cd 3dcitydb-2_0_5/oracle_spatial/UTIL/RO_USER/
[oracle@ssd RO_USER]$ ls
CREATE_RO_USER.sql
[oracle@ssd RO_USER]$ sqlplus sys as sysdba

SQL*Plus: Release 10.2.0.4.0 - Production on Tue May 24 15:27:34 2011

Copyright (c) 1982, 2007, Oracle. All Rights Reserved.

Enter password:

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> @CREATE_RO_USER

Please enter a username for the read-only user: guest
Please enter the owner of the schema to which this user will have read-only access: rotterdam

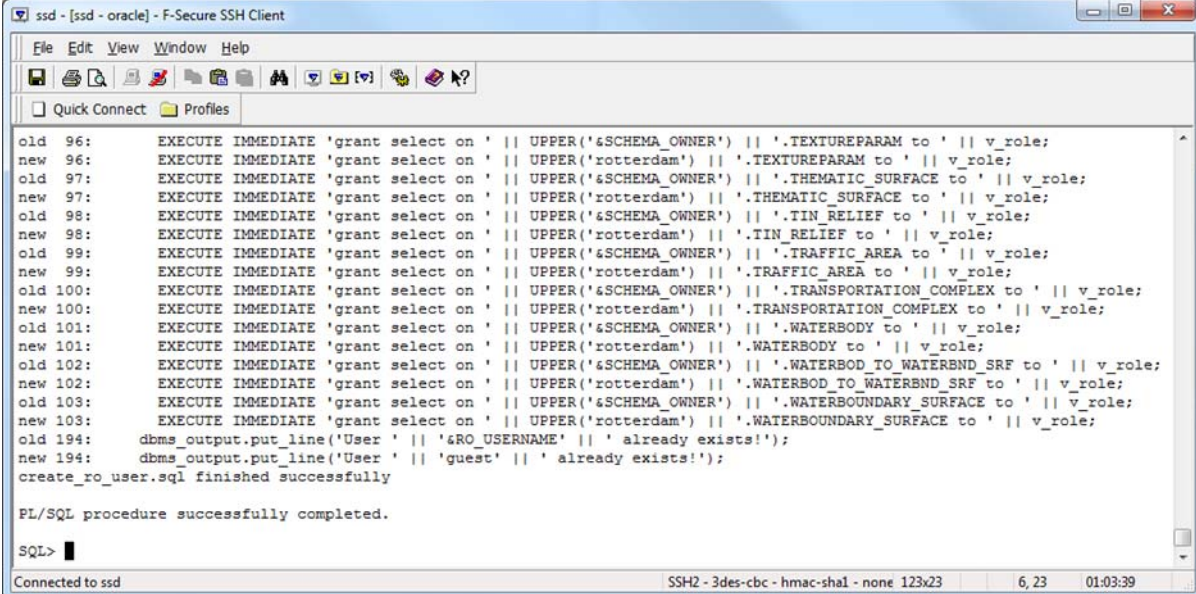
Connected to ssd
```

Fig. 23: Entering the data for a new read-only user.

5. You will be asked to enter the name of the new read-only user to be created. This user

must not exist in the database yet. Then you will be asked to enter the name of the owner of the schema (the user actually holding the city model data) to which this user shall have read-only access to. The owner of the schema *must* exist in the database prior to executing this script.

6. The read-only user will be created and the script will exit informing you of whether it finished successfully or not.



```

old 96: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.TEXTUREPARAM to ' || v_role;
new 96: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.TEXTUREPARAM to ' || v_role;
old 97: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.THEMATIC_SURFACE to ' || v_role;
new 97: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.THEMATIC_SURFACE to ' || v_role;
old 98: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.TIN_RELIEF to ' || v_role;
new 98: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.TIN_RELIEF to ' || v_role;
old 99: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.TRAFFIC_AREA to ' || v_role;
new 99: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.TRAFFIC_AREA to ' || v_role;
old 100: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.TRANSPORTATION_COMPLEX to ' || v_role;
new 100: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.TRANSPORTATION_COMPLEX to ' || v_role;
old 101: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.WATERBODY to ' || v_role;
new 101: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.WATERBODY to ' || v_role;
old 102: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.WATERBOD_TO_WATERBND_SRF to ' || v_role;
new 102: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.WATERBOD_TO_WATERBND_SRF to ' || v_role;
old 103: EXECUTE IMMEDIATE 'grant select on ' || UPPER('&SCHEMA_OWNER') || '.WATERBOUNDARY_SURFACE to ' || v_role;
new 103: EXECUTE IMMEDIATE 'grant select on ' || UPPER('rotterdam') || '.WATERBOUNDARY_SURFACE to ' || v_role;
old 194: dbms_output.put_line('User ' || '&RO_USERNAME' || ' already exists!');
new 194: dbms_output.put_line('User ' || 'guest' || ' already exists!');
create_ro_user.sql finished successfully

PL/SQL procedure successfully completed.

SQL>

```

Connected to ssd SSH2 - 3des-cbc - hmac-sha1 - none 123x23 6, 23 01:03:39

Fig. 24: Read-only user was successfully created.

7. The newly created read-only user will have a default expired password "berlin3d" (without quotes). You must log in at least once with this password and change it to a new valid one for this user to be able to do some work on the 3DCityDB.
8. You can now start the Import/Export Tool, define a database connection on the Database tab with this user's data, and start working (CityGML and KML/COLLADA exports only).

Note: Read-only users created with older versions of the 3DCityDB (2.0.3) will not work on databases upgraded to 2.0.5. Old read-only users must be dropped (no data will be lost in the process since read-only users hold none) and created again with the 2.0.5 version of the CREATE_RO_USER script.

3.7 Test data and template files

The Importer/Exporter is shipped with a set of sample data comprising:

- **CityGML instance documents**

Included are several CityGML instance documents which facilitate the testing of the CityGML import and export functionality as well as the newly introduced KML/COLLADA export capabilities.

- **KML/COLLADA models**

Sample KML/COLLADA models are provided which were created using the new KML/COLLADA export feature. The models exemplify many of the possibilities offered by this new tool. They are ready for use and exploration in any viewer application supporting the KML and COLLADA formats, but have especially been tested for use with the earth browser Google Earth.

- **KML balloon template files**

One feature of the new KML/COLLADA exporter is the creation of KML information balloons which are associated with the exported building objects. User-defined template files determining the layout and contents of the balloons can be fed to the Importer/Exporter. A generic query language allows for filling the balloon templates with object-related information at export time. The Importer/Exporter is shipped with example balloon templates demonstrating this new feature.

- **Coordinate Reference System template files**

A non-exhaustive set of CRS templates which are provided in separate files and, thus, can be easily imported and added to the list of user-defined CRSs in the Importer/Exporter. The URN encoding of these predefined CRSs generally lacks a height reference system which has to be added before using this CRS as target reference system for CityGML exports (cf. chapter 3.2 for more information).

The contained CityGML and KML/COLLADA datasets can be found in the subfolders `samples/CityGML` respectively `samples/KML_COLLADA` within the installation folder of the Importer/Exporter. The datasets are optional and will only be copied to this location if the corresponding option is selected in the setup wizard of the Importer/Exporter during the installation procedure (by default, the sample datasets will be installed). The datasets require approx. 100MB of additional hard disk space.

The KML balloon template files are located in the subfolder `templates/balloons` of the installation folder. They will always be available after installation of the Importer/Exporter and may serve as a starting point for attaching information bubbles to your KML/COLLADA models. Finally, the CRS template files are organized in subfolders beneath the `templates/CoordinateReferenceSystems` folder located in the installation folder.

Further information and descriptions of the sample test data as well as the KML balloon template files are provided in additional README files.

4 Requirements

This chapter provides an overview of the minimum requirements for the 3D City Database and the Importer/Exporter tool. Please carefully review these requirements. Additional information can be found in the previous documentation of the 3D City Database version 2.0.1 (cf. [1]).

3D City Database

The 3D City Database requires access to a working installation of one of the following Oracle Spatial DBMSs:

- Oracle Spatial 10g R2,
- Oracle Spatial 11g R1, or
- Oracle Spatial 11g R2.

Installation of all available patches from Oracle is highly recommended for optimal stability and performance. For Oracle 10g R2, at least patch set 10.2.0.4.0 is required for using the Matching/Merging tool or the newly introduced KML/COLLADA export capabilities.

Make sure to use the SQL scripts v2.0.5 shipped with this release for setting up a new instance of the 3D City Database on top of your Oracle Spatial DBMS. In case an existing 3D City Database instance of any previous version (2.0.3 or lower) shall be upgraded, please stick to the upgrade guidelines given in chapter 5.

Importer/Exporter Tool

The Importer/Exporter tool can run on any platform providing support for Java 6. It has been successfully tested on (but is not limited to) the following operating systems: Microsoft Windows XP, Vista, 7; Apple Mac OS X 10.6; Ubuntu 9, 10, 11.

Prior to the setup of the Importer/Exporter tool, the Java 6 Runtime Environment (JRE version 1.6.0_05 or higher) must be installed on your system. The necessary installation package can be obtained from <http://www.java.com/de/download>.

The Importer/Exporter tool 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. 16 MB of hard disk space. Installation packages can be chosen during the setup process.

The Importer/Exporter and requires at least 256 MB of main memory. For the import and export of large CityGML respectively KML/COLLADA files, a minimum of 1 GB of main memory is recommended.

Please note that Importer/Exporter v1.3.0 has a **mandatory dependency** on the 3D City Database v2.0.5. Read more about this in the database upgrade guidelines given in the following chapter 5.

5 Upgrade from previous versions of the 3D City Database

Older versions of the 3D City Database (version 2.0.0 to 2.0.3) **must be upgraded to the current version 2.0.5** in order to include the latest bug fixes and make use of the new features included in this release of the Importer/Exporter tool.

Note: The Importer/Exporter tool will still be able to connect to previous versions of the 3D City Database, but it will not be possible to make use of the newly introduced features on these previous versions and unexpected behavior and errors may occur. Likewise, avoid connecting with previous versions of the Importer/Exporter to a 3D City Database version 2.0.5 instance. Again, a connection will be possible but unexpected behavior may occur which may lead to severe database errors.

The 3D City Database 2.0.5 SQL package includes an upgrade script which will automatically perform the database upgrade. Upgrades to 2.0.5 can be carried out from any older version 2.0.0 to 2.0.3 (2.0.4 was an internal build only and has not been released for the public). No intermediate upgrades between lower versions are necessary.

Note: Upgrading does neither change the existing table structure nor does it affect the data stored in the tables. The upgrade procedure can thus be applied to a running database instance. However, it is recommended to first apply the upgrade in a testing environment before upgrading a production system. Furthermore, a database backup is recommended to secure all data. The latter can be easily done using the Importer/Exporter tool or by tools provided in the Oracle DBMS package.

Changes carried out in the upgrade procedure

1. *Uninstallation of the previous version of the GEODB PL/SQL package*

The GEODB PL/SQL package is part of each release of the 3D City Database. It contains several subpackages offering a range of functionality for use with the 3D City Database. The Importer/Exporter relies on these packages and they can also be used by third-party applications. With this release, some of these packages have been reworked and new packages have been added. The upgrade script will thus cleanly remove any previous version of the GEODB package.

2. *Installation of the GEODB PL/SQL package version 2.0.5*

Since the latest version 2.0.5 of the GEODB PL/SQL package is a mandatory dependency of the Importer/Exporter version 1.3.0, the upgrade script will install this versions in a second step.

3. *Renaming of indexes and constraints*

Oracle allows a maximum of 30 characters for index and constraint names on tables. If these tables are version-enabled, one has to make sure that the names do not exceed a maximum of 26 characters since Oracle internally appends suffixes required for the

version management. However, some of the index and constraint names of previous versions of the 3D City Database are longer than 26 characters. This causes errors in some functions of Oracle's Workspace Manager if versioning support is enabled. To overcome this problem, the affected names are shortened in the third step of the upgrade procedure.

4. *Creation of additional indexes*

With the release of version 2.0.2 of the 3D City Database, two additional indexes were introduced on the table `APPEAR_TO_SURFACE_DATA` which help to improve performance when exporting fully textured models from the database. The upgrade script makes sure to add these two indexes in a last step if they are missing.

Step by step guide through the upgrade procedure

The distribution package of the 3D City Database version 2.0.5 contains all SQL scripts required for setting up a new instance of the 3D City Database or for upgrading from previous versions. The package can be downloaded as ZIP archive from the 3D City Database website at <http://www.3dcitydb.net>. Alternatively, the setup wizard of the Importer/Exporter offers the possibility to copy the contents of this distribution package to the installation folder of the Importer/Exporter (simply choose the corresponding installation package in the setup procedure). In the latter case, the scripts are located in the subfolder `3dcitydb`.

1. Open a console window.
2. Change the working directory to the folder where the contents of the 3D City Database distribution package are located. In that folder, change to the `UPGRADES/2.0.5` subfolder. There is a single file in this subfolder called **`UPGRADE_DB_TO_2_0_5.sql`**. This is the main SQL script controlling the upgrade procedure. Besides this script, a further subfolder called `SCRIPTS` is located here. The `SCRIPTS` folder contains additional SQL scripts which perform single steps of the upgrade procedure and which are automatically executed in the background.

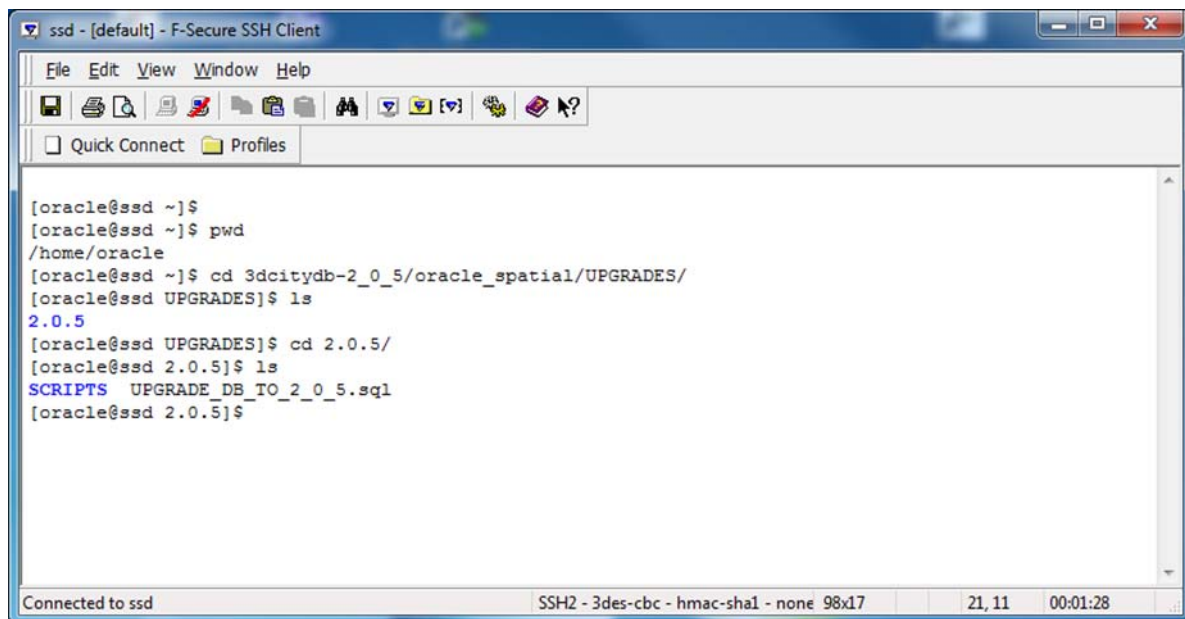


Fig. 25: Preparing for DB upgrade.

3. Open a SQLPlus connection to the account of the 3D City Database which shall be upgraded. Any SQLPlus client software may be used.
4. Execute the script `UPGRADE_DB_TO_2_0_5.sql` by typing `"@UPGRADE_DB_TO_2_0_5;"` at the SQLPlus prompt. The upgrade will immediately start and automatically execute the above described upgrading steps. An upgrade protocol is printed onto the screen (see below).

Note: Make sure to change to the `UPGRADES/2.0.5` folder *before* running SQLPlus and executing this command.

5. You will be informed about conclusion of the upgrading progress by the message "DB upgrade complete!" and the returning of the prompt.

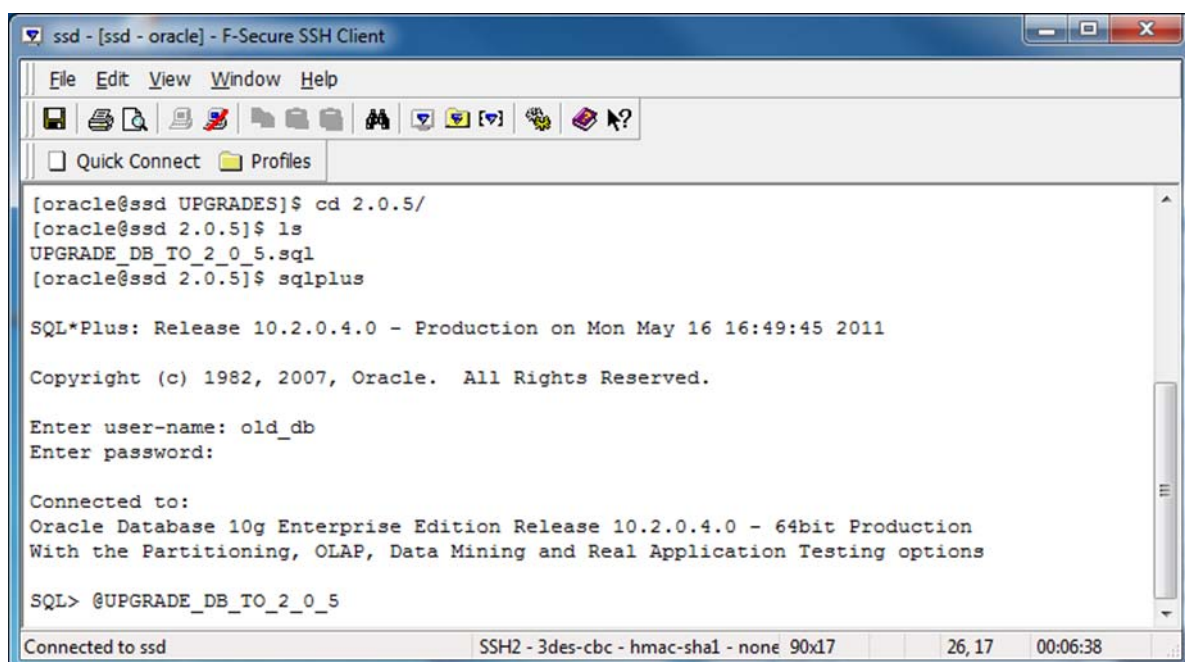


Fig. 26: Launching the upgrade script.

```

SQL> @UPGRADE_DB_TO_2_0_5;

MESSAGE
-----
Starting DB upgrade...
Starting GEODB package deletion...

Type STRARRAY deleted
Type INDEX_OBJ deleted
Package geodb_util deleted
Package geodb_idx deleted
Package geodb_stat deleted
Package geodb_match deleted
Package geodb_process_matches deleted
Package geodb_delete_by_lineage deleted
Table match_result deleted
Table match_master_aggr_geom deleted
Table match_cand_aggr_geom deleted
Table match_allocate_geom deleted
Table match_tmp_building truncated
Table match_tmp_building deleted
Table match_result_relevant truncated
Table match_result_relevant deleted
Table collect_geom deleted
Table container_ids deleted

GEODB package deletion complete!
PL/SQL procedure successfully completed.

MESSAGE
-----
Creating packages 'geodb_util', 'geodb_idx', 'geodb_stat', 'geodb_delete_by_lineage', 'geodb_delete', and corresponding types

Package created.
Package body created.
Package created.
Package body created.

MESSAGE
-----
Packages 'geodb_match', and 'geodb_merge' created

MESSAGE
-----
Starting renaming of index and constraint names. This may take a while...

Renaming APPEARANCE_TO_SURFACE_PK on table APPEAR_TO_SURFACE_DATA
Renaming RELIEF_FEATURE_TO_RELIEF_PK on table RELIEF_FEAT_TO_REL_COMP
Renaming CITYOBJECT_GENERICATTRIB_PK on table CITYOBJECT_GENERICATTRIB
Renaming SOLITARY_VEGETATION_OBJECT_PK on table SOLITARY_VEGETAT_OBJECT
Renaming CITYOBJECT_GENERICATTRIB_FKX on table CITYOBJECT_GENERICATTRIB
Renaming CITYOBJECT_GENERICATTRIB_FKX1 on table CITYOBJECT_GENERICATTRIB
Renaming SOLITARY_VEGETAT_OBJECT_FKX1 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX2 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX3 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX4 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX5 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX6 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX7 on table SOLITARY_VEGETAT_OBJECT
Renaming SOLITARY_VEGETAT_OBJECT_FKX8 on table SOLITARY_VEGETAT_OBJECT
Renaming TRANSPORTATION_COMPLEX_FKX1 on table TRANSPORTATION_COMPLEX
Renaming TRANSPORTATION_COMPLEX_FKX2 on table TRANSPORTATION_COMPLEX
Renaming TRANSPORTATION_COMPLEX_FKX3 on table TRANSPORTATION_COMPLEX
Renaming TRANSPORTATION_COMPLEX_FKX4 on table TRANSPORTATION_COMPLEX
Renaming BUILDING_FURN_LOD4REFPNT_SPX on table BUILDING_FURNITURE
Renaming SOL_VEGETAT_OBJ_LOD1REFPNT_SPX on table SOLITARY_VEGETAT_OBJECT
Renaming SOL_VEGETAT_OBJ_LOD2REFPNT_SPX on table SOLITARY_VEGETAT_OBJECT
Renaming SOL_VEGETAT_OBJ_LOD3REFPNT_SPX on table SOLITARY_VEGETAT_OBJECT
Renaming SOL_VEGETAT_OBJ_LOD4REFPNT_SPX on table SOLITARY_VEGETAT_OBJECT
rename_objects.sql successfully finished.

MESSAGE
-----
Creating additional indexes on APPEAR_TO_SURFACE_DATA. This may take a while...
Creating index APP_TO_SURF_DATA_FKX.
Creating index APP_TO_SURF_DATA_FKX1.

MESSAGE
-----
DB upgrade complete!

```


6 Changelog

The most notable changes and bug fixes from the previous versions of both the 3D City Database and the Import/Export tool are listed in the following sections. The tables comprise the revision number and a brief description of the change or bug fix.

The software development progress can be followed at any time at <http://opportunity.bv.tu-berlin.de/software>. This website also provides changelogs for previous releases of the 3D City Database and the Importer/Exporter tool.

6.1 Changelog for the 3D City Database

Please visit <http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb/repository> for a complete revision overview of the 3D City Database development and access to the source code.

Notable changes between 2.0.3 to 2.0.5	
Revision	Description
r78	Reworked script for creating read-only users to properly work with version 2.0.5.
r74	Added upgrade scripts allowing for upgrading from any previous version of the 3D City Database (2.0.3 or lower) to version 2.0.5.
r43 , r62 , r66	Major rework of the matching/merging functionality; matching/merging now supports both Oracle 10g and 11g.
r39	Added first version of GEODB_DELETE package which allows for the deletion of single objects in the database.
r33 , r34	Reworked GEODB_IDX package (providing helper procedures for spatial and non-spatial index handling) to correctly work with version-enabled tables.

Bug fixes from version 2.0.3 to 2.0.5	
Revision	Description
r72	Fixed PlanningManager compilation errors during database creation.
r38	Fixed bugs in DELETE_BY_LINEAGE.sql which prevented the script from working on versioning enabled tables.
r37	Fixed index/constraint names having more than 26 characters which cause problems on version-enabled tables.
r35	Fixed bug in database report generator script (GEODB_STAT package).

6.2 Changelog for the Importer/Exporter

Please visit <http://opportunity.bv.tu-berlin.de/software/projects/3dcitydb-imp-exp/repository> for a complete revision overview of the 3D City Database Importer/Exporter development and access to the source code. Due to a server crash, the version control system had to be reinitialized on 2011-03-07. Changes and bug fixes before that date are not listed in the current version control system anymore, and thus are given with their date in the following tables.

Notable changes between 1.2.2 to 1.3.0	
Revision / Date	Description
r157	Added standard editing popup menu for cut/copy/paste actions to input fields.
r11	Introduced information about cooperation partners into GUI.
2011-03-04	First integration of 3DCityDB scripts version 2.0.5.
2011-02-26	Major rework of matching/merging PL/SQL functionality.
2010-12-14	Added coordinate transformation to user-defined CRS for CityGML exports.
2010-11-09	Support for the generation of texture atlases during KML/COLLADA exports (outsourced in library textureAtlas.jar).
2010-10-15	Added support and management of user-defined CRS.
2010-07-01	Inaccessible options are automatically disabled in the GUI.
2010-06-28	Calculation of bounding boxes for different CityGML feature types and different coordinate reference systems.
2010-06-04	Tiling can now be applied to CityGML exports.
2010-05-20	First integration of KML/COLLADA export functionality.

Bug fixes from version 1.2.2 to 1.3.0	
Revision / Date	Description
r205	Fixed problems with wrong coordinate values of interior rings of polygons which are referenced by an XLink and which have to be reversed due to an OrientableSurface at the same time.
r193	Fixed bug when importing/exporting OrientableSurface elements having textures: The order of texture coordinates was not consistent with the order of coordinate tuples in case the orientation of the surface had to be flipped.
r154	Texture image URIs containing backslashes as path separator were not correctly interpreted on UNIX/LINUX machines.
r130 , r153	Fixed bug when importing 0 byte texture files.
r114	Changed suffix for artificial gml:ids on gml:LinearRing elements in order to create unambiguous IDs.
r45	Added batch counter to all CityGML importer classes in order to not exceed Oracle's predefined maximum size for batch updates.

2011-03-01	Adapted some SQL queries for faster execution on Oracle 11g
2011-02-18	Fixed missing support for gml:CompositeSolid geometries when exporting Building, Room, and WaterBody features.
2011-02-08	Minor fixes for import of deprecated CityGML TexturedSurface elements.
2011-01-07	Database connection pool did not correctly handle failed database connections.
2011-01-07, r42	Fixed Oracle deadlock exceptions caused by update statements in batch mode (especially when resolving XLinks).
2010-12-14, r26	Fixed UTF8 issues in GUI and log console.
2010-12-06	Solid geometries of rooms (lod4Solid property) now reference shared _BoundarySurface geometries - and not vice versa.
2010-10-21	Fixed bugs with importing ImplicitGeometry instances.
2010-09-30	Fixed bug when exporting appearances with different themes for the same city object.
2010-07-06	Fixed NPE in DBExporterManager.java.
2010-07-05	Improved memory consumption when importing large CityGML top-level features.
2010-07-01	Added support for <pointMembers> in MassPointRelief.
2010-06-29	Added workspace dialog to matching/merging panel.

7 References

- [1] Kolbe, T. H.; König, G.; Nagel, C.; Stadler, A. (2009): *3D-Geo-Database for CityGML*, Version 2.0.1, Documentation, April 24th.
http://opportunity.bv.tu-berlin.de/software/attachments/606/3DCityDB-Documentation-v2_0.pdf
- [2] 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
- [3] 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
- [4] The Google Elevation API, Weblink (accessed May 2011):
<http://code.google.com/intl/en/apis/maps/documentation/elevation/>
- [5] Active Server Pages Reference, Weblink (accessed May 2011):
<http://msdn.microsoft.com/en-us/library/ms526064.aspx>
- [6] 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
- [7] java - the Java application launcher, Weblink (accessed May 2011):
<http://download.oracle.com/javase/6/docs/technotes/tools/windows/java.html>
- [8] British German Academic Research Collaboration Programme on *Efficient Algorithms for 2-D Rectangle packing*. Weblink (accessed May 2011):
<http://www.csc.liv.ac.uk/~epa/surveyhtml.html>
- [9] 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.
- [10] 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.
- [11] D. Sleator (1980): *A 2.5 times optimal algorithm for packing in two dimensions*. In: *Information Processing Letters*, Volume 10, Number 1, pp. 37–40.
- [12] E.G. Coffman Jr., M.R. Garey, D.S. Johnson, R.E. Tarjan (1980): *Performance bounds for level-oriented two-dimensional packing algorithms*. In: *SIAM Journal on Computing* 9 (1980), pp. 801–826.

- [13] 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