

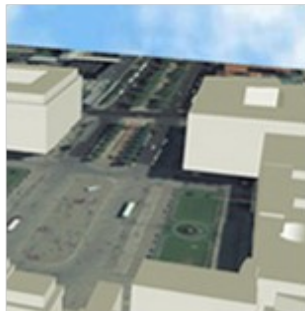
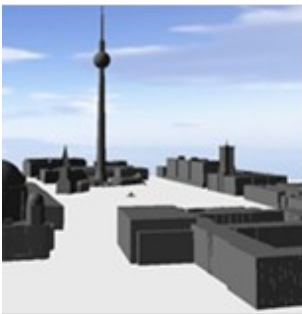
# 3D City Database for CityGML

3D City Database Version 2.0.6-postgis  
Importer/Exporter Version 1.4.0-postgis

**Release Version**

**Port documentation: Java**

**27 August 2012**



**Geoinformation Research Group  
Department of Geography  
University of Potsdam**

Felix Kunde  
Hartmut Asche

**Institute for Geodesy and  
Geoinformation Science  
Technische Universität Berlin**

Thomas H. Kolbe  
Claus Nagel  
Javier Herruela  
Gerhard König



(Page intentionally left blank)

# Content:

0	Legend.....	<b>4</b>
1	Connection to the database.....	<b>5</b>
2	Calling the PL/pgSQL-Functions.....	<b>6</b>
2.1	index-functions, datenbase-report, utility-functions inside of SQL statements.....	7
2.2	Calculation of the BoundingBox.....	8
3	Database specifics in Java.....	<b>8</b>
3.1	The Database-SRS.....	10
3.2	BoundingBox-filter and Optimizer Hints in DBSplitter.java.....	11
3.3	Queries for import.....	12
3.4	Create Table without "nologging".....	13
3.5	Data types in cached tables.....	13
4	Implicit sequences.....	<b>14</b>
5	How to work with database geometries in Java.....	<b>15</b>
5.1	From CityGML to 3D-CtyDB.....	15
5.2	From 3DCityDB back to CityGML.....	20
5.3	Synchronization of geometric functions.....	26
6	How to deal with textures.....	<b>27</b>
6.1	Import of textures.....	27
6.2	Export of textures.....	29
7	The batchsize of PostgreSQL.....	<b>31</b>
8	Workspace-Management.....	<b>32</b>
9	KML-Exporter.....	<b>33</b>
9.1	Queries.....	33
9.2	Geometries for KML placemarks.....	39
9.3	Textures for COLLADA-Export.....	44

Welcome to the documentation about ported java-classes for the *PostGIS* version of the *Importer/Exporter* tool. This document only shows exemplary parts of classes that hold database-specific Java code. Even though they are of a large number the software works mostly database-independant and had not been changed too much in the end. This documentation is divided into thematic parts and not in software packages. Info boxes at the start of each chapter should provide a quick overview which classes had to be changed and which packages were affected by this.

## 0. Legend

### Packages:

- ☐ api      = no classes in this package were changed
- ☒ database   = some parts of this package were changed
- ☒ modules    = package contains parts which need to be translated in the future

### Location of classes:

[A]	from package api	[M cityC]	modules.citygml.common
[Cmd]	cmd	[M cityE]	modules.citygml.exporter
[C]	config	[M cityI]	modules.citygml.importer
[D]	database	[M com]	modules.common
[E]	event	[M db]	modules.database
[G]	gui	[M kml]	modules.kml
[L]	log	[M pref]	modules.preferences
[P]	plugin	[oracle]	oracle.spatial.geometry
[U]	util		

### Code:

- 59**      changes start at line 59 in the corresponding class
  - 115+**   these lines could not be translated but were also not necessary in function
  - rep**    this code-example is repeating itself in the same class
  - rep+**   this code-example is repeating itself in the same class and in other classes
- 
- |   |  |
|---|--|
| <code>//private Integer port = 1521;</code> | uncommented <i>Oracle</i> -specific code<br>(already deleted from the classes) |
| <code>private Integer port = 5432;</code>   | <i>PostGIS</i> -specific code  |

## 1. Connection to the Database

Packages:	Classes:
<input type="checkbox"/> api	[Cmd] ImpExpCmd
<input checked="" type="checkbox"/> cmd	[C] DBConnection
<input checked="" type="checkbox"/> config	[D] DatabaseConnectionPool
<input checked="" type="checkbox"/> database	[D] DatabaseControllerImpl
<input type="checkbox"/> event	[M cityC] BranchTemporaryCacheTable
<input checked="" type="checkbox"/> gui	[M cityC] CacheManager
<input type="checkbox"/> log	[M cityC] HeapCacheTable
<input checked="" type="checkbox"/> modules	[M cityC] TemporaryCacheTable
<input checked="" type="checkbox"/> plugin	[M cityE] DBExportWorker
<input checked="" type="checkbox"/> util	[M cityE] DBExportWorkerFactory
	[M cityE] DBXlinkWorker
	[M cityE] DBXlinkWorkerFactory
	[M cityE] Exporter
	[M cityE] DBSplitter
	[M cityE] ExportPanel
	[M cityI] DBImportWorker
	[M cityI] DBImportWorkerFactory
	[M cityI] DBImportXlinkResolverWorker
	[M cityI] DBImportXlinkResolverWorkerFactory
	[M cityI] Importer
	[M cityI] DBCityObject
	[M cityI] DBStGeometry
	[M cityI] DBSurfaceData
	[M cityI] DBSurfaceGeometry
	[M cityI] XlinkWorldFile
	[M cityI] ImportPanel
	[M com] BoundingBoxFilter
	[M db] SrsPanel
	[G] ImpExpGui
	[G] SrsComboBoxFactory
	[P] IllegalPluginEventChecker
	[U] DBUtil

Connection handling has not changed much for the *PostgreSQL* database because the *Universal Connection Pool (UCP)* by Oracle is still used. The `PoolDataSource` of the *UCP* must pool a proper `DataSource` of *PostgreSQL*. If using the class `PGSimpleDataSource`, the URL which usually addresses the JDBC driver of an DBMS won't work properly as the result of `conn.getSid()` is not interpreted as the actual database name. To work within a network the server-name and the port-number would need to be set as well. Therefore the `org.postgresql.Driver` class was chosen in order to be able to use a connection URL. Connection-properties were uncommented as the `PGConnection` class of *PostgreSQL* only holds the same attributes than the Java `Connection` class. `CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE` was not offered.

Unfortunately the use of Oracle's *UCP* is not conform to the OpenSource effort behind the *PostGIS* version of the *3DCityDB*. The Apache *Jakarta DBCP* was tested by the developers but found to work unacceptably worse than the *UCP*. The Connection Pools of Apache's *Tomcat 7* or *C3PO* should be an alternative. As seen by the number of orange packages in the overview-box, this would mean a lot of code rework.

de.tub.citydb.config.project.database.**DBConnection**

```
59    //private Integer port = 1521;
    private Integer port = 5432;
```

de.tub.citydb.database.**DatabaseConnectionPool**

```
64    //private final String poolName = "oracle.pool";
    private final String poolName = "postgresql.pool";

115    // poolDataSource.setConnectionFactoryClassName (
    //      "oracle.jdbc.pool.OracleDataSource");
    //
    // poolDataSource.setURL("jdbc:oracle:thin:@/" + conn.getServer() + ":" +
    //      conn.getPort() + "/" + conn.getSid());
    poolDataSource.setConnectionFactoryClassName("org.postgresql.Driver");
    poolDataSource.setURL("jdbc:postgresql://" + conn.getServer() + ":" +
    //      conn.getPort() + "/" + conn.getSid());

115+    // set connection properties
```

## 2. Calling the PL/pgSQL-functions

Packages:	Classes:
<input type="checkbox"/> api	[M db] IndexOperation
<input type="checkbox"/> cmd	[M cityI] Importer
<input type="checkbox"/> config	[M cityE] DBAppearance
<input type="checkbox"/> database	[M cityE] DBBuilding
<input type="checkbox"/> event	[M cityE] DBBuildingFurniture
<input type="checkbox"/> gui	[M cityE] DBCityFurniture
<input type="checkbox"/> log	[M cityE] DBCityObject
<input type="checkbox"/> modules	[M cityE] DBCityObjectGroup
<input type="checkbox"/> plugin	[M cityE] DBGeneralization
<input checked="" type="checkbox"/> util	[M cityE] DBGenericCityObject
	[M cityE] DBReliefFeature
	[M cityE] DBSolitaryVegetatObject
	[M cityE] DBSurfaceGeometry
	[M cityE] DBThematicSurface
	[M cityE] DBTransportationComplex
	[M cityE] DBWaterBody
	[U] DBUtil

Most of the functionalities in the database panel of the *Importer/Exporter* are calling stored procedures in the database. So the main changes in code were done in the PL/pgSQL scripts. Within Java only the names of the called functions were changed. The functions are bundled inside of a database-schema called "geodb\_pkg".

## 2.1 index-functions, database-report, utility-functions inside of statements

The bigger the size of files to be imported the longer it takes to index the data after every inserted tuple. It is recommended to drop the indexes before importing data and recreate them afterwards. *Oracle* keeps metadata of a dropped index, *PostgreSQL* does not. An alternative way was programmed but it is not used now. The idea was to just set the index-status to invalid (pg\_index.indisvalid) that it stays inactive during the import and then REINDEX it afterwards. It was only tested with small datasets but no performance improvement could be detected. The functions are already written but they are not a part of the recent release.

für alle de.tub.citydb.modules.citygml.exporter.database.content.**DB\***

```
//geodb_util.transform_or_null(...)
geodb_pkg.util_transform_or_null(...
```

de.tub.citydb.util.database.**DBUtil**

```
73  // private static OracleCallableStatement callableStmt;
    private static CallableStatement callableStmt;

91  // rs = stmt.executeQuery("select * from table(geodb_util.db_metadata)");
    rs = stmt.executeQuery("select * from geodb_pkg.util_db_metadata() as t");

199 // callableStmt = (OracleCallableStatement)conn.prepareCall("{? = call
rep //   geodb_stat.table_contents}");
    callableStmt = (CallableStatement)conn.prepareCall("{? = call
        geodb_pkg.stat_table_contents()}");

200 // callableStmt.registerOutParameter(1, OracleTypes.ARRAY, "STRARRAY");
rep callableStmt.registerOutParameter(1, Types.ARRAY);

203 // ARRAY result = callableStmt.getARRAY(1);
rep Array result = callableStmt.getArray(1);

374 // String call = type == DBIndexType.SPATIAL ?
rep //     "{? = call geodb_idx.drop_spatial_indexes}" :
    //     "{? = call geodb_idx.drop_normal_indexes}";
Drop Case:
String call = type == DBIndexType.SPATIAL ?
    "{? = call geodb_pkg.idx_drop_spatial_indexes}" :
    "{? = call geodb_pkg.idx_drop_normal_indexes}";
or Switch-Case:
String call = type == DBIndexType.SPATIAL ?
    "{? = call geodb_pkg.idx_switch_off_spatial_indexes}" :
    "{? = call geodb_pkg.idx_switch_off_normal_indexes}";
// callableStmt = (OracleCallableStatement)conn.prepareCall(call);
callableStmt = (CallableStatement)conn.prepareCall(call);
```

## 2.2 Calculation of the BoundingBox

For the calculation of the BoundingBox workspace-variables were uncommented. The query strings had to call equivalent *PostGIS* functions (e.g. `sdo_aggr_mbr --> ST_Extent`, `geodb_util.to2d --> ST_Force_2d`). As rectangle geometries can not be shorten in number of points like in *Oracle* (LLB, URT), 5 Points were needed for the coordinate-transformation. As placeholders for single coordinates did not work with a `PreparedStatement` the whole String in the *PostGIS* function `ST_GeomFromEWKT(?)` was used as the exchangeable variable.

de.tub.citydb.util.database.**DBUtil**

```

237 // public static BoundingBox calcBoundingBox(Workspace workspace,
//      FeatureClassMode featureClass) throws SQLException {
public static BoundingBox calcBoundingBox(FeatureClassMode featureClass)
    throws SQLException {

248 // String query = "select sdo_aggr_mbr(geodb_util.to_2d(
//      ENVELOPE, (select srid from database_srs)))
//      from CITYOBJECT where ENVELOPE is not NULL";
String query = "select ST_Extent(ST_Force_2d(envelope))::geometry
    from cityobject where envelope is not null";

317 // double[] points = jGeom.getOrdinatesArray();
// if (dim == 2) {
//     xmin = points[0];
//     ymin = points[1];
//     xmax = points[2];
//     ymax = points[3];
// } else if (dim == 3) {
//     xmin = points[0];
//     ymin = points[1];
//     xmax = points[3];
//     ymax = points[4];
// }
xmin = (geom.getPoint(0).x);
ymin = (geom.getPoint(0).y);
xmax = (geom.getPoint(2).x);
ymax = (geom.getPoint(2).y);

629 // psQuery = conn.prepareStatement("select SDO_CS.TRANSFORM(
//      MDSYS.SDO_GEOMETRY(2003, " + sourceSrid + ", NULL,
//      MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 1), " +
//      "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), " + targetSrid + ")from dual");
// psQuery.setDouble(1, bbox.getLowerLeftCorner().getX());
// psQuery.setDouble(2, bbox.getLowerLeftCorner().getY());
// psQuery.setDouble(3, bbox.getUpperRightCorner().getX());
// psQuery.setDouble(4, bbox.getUpperRightCorner().getY());
psQuery = conn.prepareStatement("select ST_Transform(ST_GeomFromEWKT(?), "
    + targetSrid + ")");

```



```

boxGeom = "SRID=" + sourceSrid + ";POLYGON((" +
    bbox.getLowerLeftCorner().getX() + " " +
    bbox.getLowerLeftCorner().getY() + "," +
    bbox.getLowerLeftCorner().getX() + " " +
    bbox.getUpperRightCorner().getY() + "," +
    bbox.getUpperRightCorner().getX() + " " +
    bbox.getUpperRightCorner().getY() + "," +
    bbox.getUpperRightCorner().getX() + " " +
    bbox.getLowerLeftCorner().getY() + "," +
    bbox.getLowerLeftCorner().getX() + " " +
    bbox.getLowerLeftCorner().getY() + "))";

```

```

psQuery.setString(1, boxGeom);

```

```

645 // double[] ordinatesArray = geom.getOrdinatesArray();
// result.getLowerCorner().setX(ordinatesArray[0]);
// result.getLowerCorner().setY(ordinatesArray[1]);
// result.getUpperCorner().setX(ordinatesArray[2]);
// result.getUpperCorner().setY(ordinatesArray[3]);
result.getLowerLeftCorner().setX(geom.getPoint(0).x);
result.getLowerLeftCorner().setY(geom.getPoint(0).y);
result.getUpperRightCorner().setX(geom.getPoint(2).x);
result.getUpperRightCorner().setY(geom.getPoint(2).y);

```

### 3. Database specifics in Java

Packages:	Classes:
<input checked="" type="checkbox"/> api	[A] DatabaseSrsType
<input type="checkbox"/> cmd	[A] DatabaseSrs
<input type="checkbox"/> config	[G] SrsComboBoxFactory
<input type="checkbox"/> database	[M cityC] CacheTableBasic
<input type="checkbox"/> event	[M cityC] CacheTableDeprecatedMaterial
<input type="checkbox"/> gui	[M cityC] CacheTableGlobalAppearance
<input type="checkbox"/> log	[M cityC] CacheTableGmllid
<input checked="" type="checkbox"/> modules	[M cityC] CacheTableGroupToCityObject
<input type="checkbox"/> plugin	[M cityC] CacheTableLibraryObject
<input checked="" type="checkbox"/> util	[M cityC] CacheTableSurfaceGeometry
	[M cityC] CacheTableTextureAssociation
	[M cityC] CacheTableTextureFile
	[M cityC] CacheTableTextureParam
	[M cityC] CacheTableModel
	[M cityC] HeapCacheTable
	[M cityE] Exporter
	[M cityE] DBAppearance
	[M cityE] DBSplitter
	[M cityI] DBCityObject
	[M cityI] DBCityObjectGenericAttrib
	[M cityI] DBExternalReference
	[M cityI] DBSequencer
	[M cityI] DBSurfaceGeometry
	[M cityI] XlinkSurfaceGeometry
	[U] DBUtil

#### 3.1 The database-SRS

Until now *PostGIS* does not offer 3D-spatial-reference-systems by default. INSERT examples for *PostGIS* can be found at [spatialreference.org](http://spatialreference.org). Unfortunately 2D and 3D geographic reference systems are equally classified as `GEOGCS`. The function `is3D()` would not detect 3D-SRIDs though. If the INSERT statement by [spatialreference.org](http://spatialreference.org) is changed manually from `GEOGCS` to `GEOGCS3D` `is3D()` would work as the type is listed in the `DatabaseSrsType` class. It is not sure how 3D-SRIDs will be handled in future *PostGIS* releases. *Oracle Spatial* has got some strict rules how to work with them. This includes certain checks on the data, which are not needed for the *PostGIS* version at the moment. It can be noticed that the `spatial_ref_sys` table in *PostGIS* contains less columns than *Oracle's* `SDO_COORD_REF_SYS`-table. Most of the information is stored in the text-column `srttext`. It can be extracted with String functions.

de.tub.citydb.api.database.**DatabaseSrsType**

```

4    PROJECTED("PROJCS", "Projected"),
    GEOGRAPHIC2D("GEOGCS", "Geographic2D"),
    GEOCENTRIC("GEOCCS", "Geocentric"),
    VERTICAL("VERT_CS", "Vertical"),
    ENGINEERING("LOCAL_CS", "Engineering"),
    COMPOUND("COMPD_CS", "Compound"),
    GEOCENTRIC("n/a", "Geogentric"),
    GEOGRAPHIC3D("GEOGCS3D", "Geographic3D"),
    UNKNOWN("", "n/a");

```

de.tub.citydb.util.database.**DBUtil**

```

141 // psQuery = conn.prepareStatement("select coord_ref_sys_name,
//   coord_ref_sys_kind from sdo_coord_ref_sys where srid = ?");
psQuery = conn.prepareStatement("select split_part(srtext, '\'', 2) as
    coord_ref_sys_name, split_part(srtext, '[', 1) as coord_ref_sys_kind
    FROM spatial_ref_sys WHERE SRID = ? ");

704 // psQuery = conn.prepareStatement(srs.getType() ==
//   DatabaseSrsType.GEOGRAPHIC3D ?
//   "select min(crs2d.srid) from sdo_coord_ref_sys crs3d,
//   sdo_coord_ref_sys crs2d where crs3d.srid = " + srs.getSrid() +
//   " and crs2d.coord_ref_sys_kind = 'GEOGRAPHIC2D'
//   and crs3d.datum_id = crs2d.datum_id" :
//   "select cmpd_horiz_srid from sdo_coord_ref_sys
//   where srid = " + srs.getSrid());
psQuery = conn.prepareStatement(srs.getType() == DatabaseSrsType.COMPOUND ?
    "select split_part((split_part(srtext, 'AUTHORITY[\"EPSG\", \"'\", 5)), '\'', 1)
    from spatial_ref_sys where auth_srid = " + srs.getSrid() :
    // searching 2D equivalent for 3D SRID
    "select min(crs2d.auth_srid) from spatial_ref_sys crs3d, spatial_ref_sys
    crs2d where (crs3d.auth_srid = " + srs.getSrid() + " and split_part
    (crs3d.srtext, '[', 1) LIKE 'GEOGCS' AND
    split_part(crs2d.srtext, '[', 1) LIKE 'GEOGCS' " +
    //do they have the same Datum_ID?
    "and split_part(
    (split_part(crs3d.srtext, 'AUTHORITY[\"EPSG\", \"'\", 3)), '\'', 1)
    = split_part(
    (split_part(crs2d.srtext, 'AUTHORITY[\"EPSG\", \"'\", 3)), '\'', 1))
    OR " +
    // if srtext has been changed for Geographic3D
    "(crs3d.auth_srid = " + srs.getSrid() + " and
    split_part(crs3d.srtext, '[', 1) LIKE 'GEOGCS3D' AND
    split_part(crs2d.srtext, '[', 1) LIKE 'GEOGCS' " +
    //do they have the same Datum_ID?
    "and split_part(
    (split_part(crs3d.srtext, 'AUTHORITY[\"EPSG\", \"'\", 3)), '\'', 1)
    = split_part(
    (split_part(crs2d.srtext, 'AUTHORITY[\"EPSG\", \"'\", 3)), '\'', 1)))");

```

### 3.2 BoundingBox-filter and OptimizerHints in DBSplitter.java

DBSplitter.java manages the filtering of data by a given bounding box. In *Oracle Spatial* the spatial operation SDO\_RELATE is used for that. SDO\_RELATE checks topological relations between geometries according to the 9-intersection Matrix (DE-9IM). It is possible to combine the mask-attributes with a logical OR (+). This is not adoptable for *PostGIS*, as the equivalent ST\_Relate operation can only use one mask. Thus the first field of the bounding box filter array contains four ST\_Relate conditions connected by "or".

Another feature of *Oracle* which is used in the DBSplitter class is the "Optimizer Hint". It is used to tell the internal query optimizer which query plan to prefer. As there are no such Optimizer Hints in *PostgreSQL* they were uncommented.

de.tub.citydb.modules.citygml.exporter.database.content.**DBSplitter**

```

168  //      String filter = "SDO_RELATE(co.ENVELOPE, MDSYS.SDO_GEOMETRY(2003, "
//          + bboxSrid + ", NULL, " +
//          "MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 3), " +
//          "MDSYS.SDO_ORDINATE_ARRAY(" + minX + ", " + minY + ", " + maxX
//          + ", " + maxY + ")), 'mask=";
//          bboxFilter[0] = filter + "inside+coveredby') = 'TRUE'";
//          bboxFilter[1] = filter + "equal') = 'TRUE'";
//      if (overlap)
//          bboxFilter[2] = filter + "overlapbdyintersect') = 'TRUE'";
String filter = "ST_Relate(co.ENVELOPE, " +
                "ST_GeomFromEWKT('SRID=" + bboxSrid + ";POLYGON((" +
                minX + " " + minY + ", " +
                minX + " " + maxY + ", " +
                maxX + " " + maxY + ", " +
                maxX + " " + minY + ", " +
                minX + " " + minY + "))'))", ";

bboxFilter[0] = "(" + filter + "'T**F**F**') = 'TRUE' or " + // inside
                filter + "'*TF**F**') = 'TRUE' or " + // coveredby
                filter + "'**FT**F**') = 'TRUE' or " + // coveredby
                filter + "'**F*TF**') = 'TRUE'"; // coveredby
bboxFilter[1] = filter + "'T**F**FF*') = 'TRUE'"; // equal

if (overlap)
    bboxFilter[2] = filter + "'T*T***T**') = 'TRUE'"; //overlapbdyinter.

```

### 3.3 Queries for the Import

Some queries of the Importer classes use *Oracle*-specific functions.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**

```

138  // nvl(sd.TEX_IMAGE.getContentLength(), 0) as DB_TEX_IMAGE_SIZE,
rep  // sd.TEX_IMAGE.getMimeType() as DB_TEX_IMAGE_MIME_TYPE, sd.TEX_MIME_TYPE,
      COALESCE(length(sd.TEX_IMAGE), 0) as DB_TEX_IMAGE_SIZE, sd.TEX_MIME_TYPE,

```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```

134  // SYSDATE
      now()

```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObjectGenericAttrib**

```

63  // CITYOBJECT_GENERICATT_SEQ.nextval
      nextval('CITYOBJECT_GENERICATTRIB_ID_SEQ')

```

de.tub.citydb.modules.citygml.importer.database.content.**DBExternalReference**

```

58  // EXTERNAL_REF_SEQ.nextval
      nextval('EXTERNAL_REFERENCE_ID_SEQ')

```

de.tub.citydb.modules.citygml.importer.database.content.**DBSequencer**

```
53    // pstmt = conn.prepareStatement("select " + sequence.toString() +  
        ".nextval from dual");  
    pstmt = conn.prepareStatement("select nextval('" + sequence.toString() +  
        "')");
```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkSurfaceGeometry**

```
92    // psSelectSurfGeom = batchConn.prepareStatement("select sg.*, LEVEL from  
        SURFACE_GEOMETRY sg start with sg.ID=? connect by prior  
        sg.ID=sg.PARENT_ID");  
    psSelectSurfGeom = batchConn.prepareStatement("WITH RECURSIVE geometry  
        (id, gmlid, gmlid_codespace, parent_id, root_id, is_solid,  
        is_composite, is_triangulated, is_xlink, is_reverse, geometry, level) "  
        " AS (SELECT sg.*, 1 AS level FROM surface_geometry sg WHERE sg.id=?  
        UNION ALL "  
        " SELECT sg.*, g.level + 1 AS level FROM  
        surface_geometry sg, geometry g WHERE sg.parent_id=g.id) " +  
        " SELECT * FROM geometry ORDER BY level asc");  
  
98    // SURFACE_GEOMETRY_SEQ.nextval  
    nextval('SURFACE_GEOMETRY_ID_SEQ')
```

### 3.4 Create Table without "nologging"

There is no `nologging` option for CREATE statements in *PostgreSQL*.

de.tub.citydb.modules.citygml.common.database.cache.model.**CacheTableModel**

```
95    // " nologging" +
```

de.tub.citydb.modules.citygml.common.database.cache.**HeapCacheTable**

```
158    model.createIndexes(conn, tableName/*, "nologging"*/);
```

### 3.5 Data types in cached tables

In the folder `common.database.cache.model` several classes had to be changed due to different data types of the DMBS. NUMBER to NUMERIC (ID-columns = integer), VARCHAR2 to VARCHAR.

## 4. Implicit sequences

Packages:	Classes:
<input type="checkbox"/> api	[M cityl] DBAddress
<input type="checkbox"/> cmd	[M cityl] DBAppearance
<input type="checkbox"/> config	[M cityl] DBBuilding
<input type="checkbox"/> database	[M cityl] DBBuildingFurniture
<input type="checkbox"/> event	[M cityl] DBBuildingInstallation
<input type="checkbox"/> gui	[M cityl] DBCityFurniture
<input type="checkbox"/> log	[M cityl] DBCityObjectGroup
<input type="checkbox"/> modules	[M cityl] DBGenericCityObject
<input type="checkbox"/> plugin	[M cityl] DBImplicitGeometry
<input type="checkbox"/> util	[M cityl] DBImporterManager
	[M cityl] DBLandUse
	[M cityl] DBOpening
	[M cityl] DBPlantCover
	[M cityl] DBReliefComponent
	[M cityl] DBReliefFeature
	[M cityl] DBRoom
	[M cityl] DBSequencerEnum
	[M cityl] DBSolitaryVegetatObject
	[M cityl] DBSurfaceData
	[M cityl] DBSurfaceGeometry
	[M cityl] DBThematicSurface
	[M cityl] DBTrafficArea
	[M cityl] DBTransportationComplex
	[M cityl] DBWaterBody
	[M cityl] DBWaterBoundarySurface
	[M cityl] XlinkDeprecatedMaterial
	[M cityl] XlinkSurfaceGeometry

In *PostgreSQL* it is quite common to assign the data type SERIAL to ID-columns which are used as primary keys. SERIAL implicitly creates a sequence with the names of table, column and the ending "\_SEQ". The declaration "CREATE SEQUENCE" must not be written manually like in *Oracle*. But this holds a trap. As names are created automatically with SERIAL they differ from the customized names in *Oracle*. See also **3.3** for examples.

de.tub.citydb.modules.citygml.importer.database.content.**DBSequencerEnum**

```

32  //public enum DBSequencerEnum {
    //    ADDRESS_SEQ,
    //    APPEARANCE_SEQ,
    //    CITYOBJECT_SEQ,
    //    SURFACE_GEOMETRY_SEQ,
    //    IMPLICIT_GEOMETRY_SEQ,
    //    SURFACE_DATA_SEQ,
    public enum DBSequencerEnum {
        ADDRESS_ID_SEQ,
        APPEARANCE_ID_SEQ,
        CITYOBJECT_ID_SEQ,
        SURFACE_GEOMETRY_ID_SEQ,
        IMPLICIT_GEOMETRY_ID_SEQ,
        SURFACE_DATA_ID_SEQ,

```

## 5. How to work with database geometries in Java

Packages:	Classes:
<input type="checkbox"/> api	[M cityE] DBAppearance
<input type="checkbox"/> cmd	[M cityE] DBBuilding
<input type="checkbox"/> config	[M cityE] DBCityFurniture
<input type="checkbox"/> database	[M cityE] DBCityObject
<input type="checkbox"/> event	[M cityE] DBGeneralization
<input type="checkbox"/> gui	[M cityE] DBGenericCityObject
<input type="checkbox"/> log	[M cityE] DBReliefFeature
<input checked="" type="checkbox"/> modules	[M cityE] DBSolitaryVegetatObject
<input type="checkbox"/> plugin	[M cityE] DBStGeometry
<input checked="" type="checkbox"/> util	[M cityE] DBSurfaceGeometry
<input checked="" type="checkbox"/> oracle.spatial.	[M cityE] DBThematicSurface
geometry	[M cityE] DBTransportationComplex
	[M cityE] DBWaterBody
	[M cityI] DBAddress
	[M cityI] DBBuilding
	[M cityI] DBBuildingFurniture
	[M cityI] DBCityFurniture
	[M cityI] DBCityObject
	[M cityI] DBGenericCityObject
	[M cityI] DBReliefComponent
	[M cityI] DBSolitaryVegetatObject
	[M cityI] DBStGeometry
	[M cityI] DBSurfaceData
	[M cityI] DBSurfaceGeometry
	[M cityI] DBTransportationComplex
	[M cityI] DBWaterBody
	[M cityI] XlinkSurfaceGeometry
	[M cityI] XlinkWorldFile
	[U] DBUtil
	[oracle] SyncJGeometry

Translating the processing of geometries to the *PostGIS* JDBC driver was with no doubt the toughest job to do. This chapter shortly explains how geometries were parsed from a CityGML document and inserted into the database and all the way back.

### 5.1 From CityGML to 3DCityDB

The *Oracle* JDBC driver handles geometries with one central class called `JGeometry`. One instance of `JGeometry` represents `SDO_GEOMETRY` in the Java world. All methods of different geometric types return `JGeometry`. They need three variables: an array of coordinates, the number of dimensions and a known SRID. The geometries of CityGML are described by geometric primitives from the `citygml4j.lib`. Their values are first transferred to list-elements and then iterated into arrays to be used by the described `JGeometry` methods. `JGeometry` can not be set as an object for the database statements. It needs to be "stored" into a `STRUCT` object, which is a wrapper-class for `JGeometry`. This wrapper makes the object more generic to be used by the `PreparedStatement` method `setObject`.

For the *PostGIS* JDBC driver the combination of geometry class and wrapper class is represented by `Geometry` and `PGGeometry`. `Geometry` offers some geometric operations, but

to create an instance of `Geometry` the `PGgeometry` method `geomFromString(String)` has to be used. The values of the list elements have to iteratively build up a string and not fill an array. The String represents the geometries in Well Known Text (WKT), which means blank spaces between coordinates (x y z) instead of commas. To be interpreted by the database the geometries have to be wrapped as a `PGgeometry` object and then set for the `PreparedStatement`.

de.tub.citydb.modules.citygml.importer.database.content.**DBAddress**

```

91    // private DBSdoGeometry sdoGeometry;
rep+ private DBStGeometry stGeometry;

106   // sdoGeometry = (DBSdoGeometry)dbImporterManager.getDBImporter(
rep+   DBImporterEnum.SDO_GEOMETRY);
    stGeometry = (DBStGeometry)dbImporterManager.getDBImporter(
        DBImporterEnum.ST_GEOMETRY);

133   // JGeometry multiPoint = null;
rep+ PGgeometry multiPoint = null;

224   // multiPoint = sdoGeometry.getMultiPoint(address.getMultiPoint());
rep+ multiPoint = stGeometry.getMultiPoint(address.getMultiPoint());

    // if (multiPoint != null) {
    //     Struct multiPointObj= SyncJGeometry.syncStore(multiPoint,batchConn);
    //     psAddress.setObject(8, multiPointObj);
    // } else
    //     psAddress.setNull(8, Types.STRUCT, "MDSYS.SDO_GEOMETRY");
    if (multiPoint != null) {
        psAddress.setObject(8, multiPoint);
    } else
        psAddress.setNull(8, Types.OTHER, "ST_GEOMETRY");

```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```

211   // double[] ordinates = new double[points.size()];
rep+ // int i = 0;
    // for (Double point : points)
    //     ordinates[i++] = point.doubleValue();
    // JGeometry boundedBy =
    //     JGeometry.createLinearPolygon(ordinates, 3, dbSrid);
    // STRUCT obj = SyncJGeometry.syncStore(boundedBy, batchConn);
    //
    // psCityObject.setObject(4, obj);
    String geomEWKT = "SRID=" + dbSrid + ";POLYGON(";
    for (int i=0; i<points.size(); i+=3){
        geomEWKT += points.get(i) + " " + points.get(i+1) + " " +
            points.get(i+2) + ",";
    }
    geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
    geomEWKT += "));";

```



```

Geometry boundedBy = PGGeometry.geomFromString(geomEWKT);
PGGeometry pgBoundedBy = new PGGeometry(boundedBy);

psCityObject.setObject(4, pgBoundedBy);

```

de.tub.citydb.modules.citygml.importer.database.content.**DBCityObject**

```

68    // SDO_GEOMETRY();
    ST_GEOMETRY();

```

de.tub.citydb.modules.citygml.importer.database.content.**DBStGeometry**

```

88    // public JGeometry getPoint(PointProperty pointProperty) {
rep    //     JGeometry pointGeom = null;
    public PGGeometry getPoint(PointProperty pointProperty) throws
    SQLException {
        Geometry pointGeom = null;

99    // double[] coords = new double[values.size()];
    // int i = 0;
    // for (Double value : values)
    //     coords[i++] = value.doubleValue();
    // pointGeom = JGeometry.createPoint(coords, 3, dbSrid);
    pointGeom = PGGeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
        values.get(0) + " " + values.get(1) + " " + values.get(2) + ")");

171 // if (!pointList.isEmpty()) {
rep //     Object[] pointArray = new Object[pointList.size()];
    //     int i = 0;
    //     for (List<Double> coordsList : pointList) {
    //         if (affineTransformation)
    //             dbImporterManager.getAffineTransformer().
    //                 transformCoordinates(coordsList);
    //
    //         double[] coords = new double[3];
    //
    //         coords[0] = coordsList.get(0).doubleValue();
    //         coords[1] = coordsList.get(1).doubleValue();
    //         coords[2] = coordsList.get(2).doubleValue();
    //
    //         pointArray[i++] = coords;
    //     }
    //     multiPointGeom = JGeometry.createMultiPoint(pointArray, 3, dbSrid);
    // }
    // }
    // return multiPointGeom;
    if (!pointList.isEmpty()) {
        String geomEWKT = "SRID=" + dbSrid + ";MULTIPOINT(";

        for (List<Double> coordsList : pointList){

            if (affineTransformation)
                dbImporterManager.getAffineTransformer().
                    transformCoordinates(coordsList);

```

```

        geomEWKT += coordsList.get(0) + " " + coordsList.get(1) + " "
                    + coordsList.get(2) + ",";
    }

    geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
    geomEWKT += " ";

    multiPointGeom = PGGeometry.geomFromString(geomEWKT);
}

}
PGGeometry pgMultiPointGeom = new PGGeometry(multiPointGeom);
return pgMultiPointGeom;

213 // if (!pointList.isEmpty()) {
rep //     Object[] pointArray = new Object[pointList.size()];
//     int i = 0;
//     for (List<Double> coordsList : pointList) {
//         if (affineTransformation)
//             dbImporterManager.getAffineTransformer().
//                 transformCoordinates(coordsList);
//         double[] coords = new double[coordsList.size()];
//         int j = 0;
//         for (Double coord : coordsList)
//             coords[j++] = coord.doubleValue();
//         pointArray[i++] = coords;
//     }
//     multiCurveGeom = JGeometry.createLinearMultiLineString(pointArray,
//         3, dbSrid);
// }
if (!pointList.isEmpty()) {
    String geomEWKT = "SRID=" + dbSrid + ";MULTILINESTRING(";

    for (List<Double> coordsList : pointList) {
        if (affineTransformation)
            dbImporterManager.getAffineTransformer().
                transformCoordinates(coordsList);

        for (int i=0; i<coordsList.size(); i+=3){
            geomEWKT += coordsList.get(i) + " " +
                coordsList.get(i+1) + " " + coordsList.get(i+2) + ",";
        }
        geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
        geomEWKT += "), (";
    }
    geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 2);
    geomEWKT += " ";
    multiCurveGeom = PGGeometry.geomFromString(geomEWKT);
}

```

de.tub.citydb.modules.citygml.importer.database.content.**DBSurfaceData**

```

437 // JGeometry geom = new JGeometry(coords.get(0), coords.get(1), dbSrid);
// STRUCT obj = SyncJGeometry.syncStore(geom, batchConn);
// psSurfaceData.setObject(15, obj);

```

```

Geometry geom = PGGeometry.geomFromString("SRID=" + dbSrid + ";POINT(" +
    coords.get(0) + " " + coords.get(1) + ")");
PGGeometry pgGeom = new PGGeometry(geom);
psSurfaceData.setObject(15, pgGeom);

```

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkSurfaceGeometry**

```

283  // if (reverse) {
//      int[] elemInfoArray = geomNode.geometry.getElemInfo();
//      double[] ordinatesArray = geomNode.geometry.getOrdinatesArray();
//
//      if (elemInfoArray.length < 3 || ordinatesArray.length == 0) {
//          geomNode.geometry = null;
//          return;
//      }
//
//      // we are pragmatic here. if elemInfoArray contains more than one
//      // entry, we suppose we have one outer ring and anything else are
//      // inner rings.
//      List<Integer> ringLimits = new ArrayList<Integer>();
//      for (int i = 3; i < elemInfoArray.length; i += 3)
//          ringLimits.add(elemInfoArray[i] - 1);
//
//      ringLimits.add(ordinatesArray.length);
//
//      // ok, reverse polygon according to this info
//      Object[] pointArray = new Object[ringLimits.size()];
//      int ringElem = 0;
//      int arrayIndex = 0;
//      for (Integer ringLimit : ringLimits) {
//          double[] coords = new double[ringLimit - ringElem];
//
//          for (int i=0, j=ringLimit-3; j>=ringElem; j-=3, i+=3) {
//              coords[i] = ordinatesArray[j];
//              coords[i + 1] = ordinatesArray[j + 1];
//              coords[i + 2] = ordinatesArray[j + 2];
//          }
//
//          pointArray[arrayIndex++] = coords;
//          ringElem = ringLimit;
//      }
//
//      JGeometry geom = JGeometry.createLinearPolygon(PointArray,
//          geomNode.geometry.getDimensions(),
//          geomNode.geometry.getSrid());
//
//      geomNode.geometry = geom;
//  }
if (reverse) {
    String geomEWKT = "SRID=" + geomNode.geometry.getSrid() +
        ";POLYGON(";
    Polygon polyGeom = (Polygon) geomNode.geometry;
    int dimensions = geomNode.geometry.getDimension();

    for (int i = 0; i < polyGeom.numRings(); i++){
        if (dimensions == 2)

```

```

        for (int j=0; j<polyGeom.getRing(i).numPoints(); j++){
            geomEWKT += polyGeom.getRing(i).getPoint(j).x + " " +
polyGeom.getRing (i).getPoint(j).y + ",";
        }

        if (dimensions == 3)
            for (int j=0; j<polyGeom.getRing (i).numPoints(); j++){
                geomEWKT += polyGeom.getRing (i).getPoint(j).x + " " +
polyGeom.getRing(i).getPoint(j).y + " " +
polyGeom.getRing(i).getPoint(j).z + ",";
            }

        geomEWKT = geomEWKT.substring(0, geomEWKT.length() - 1);
        geomEWKT += "), (";

    }

    Geometry geom = PGgeometry.geomFromString(geomEWKT);
    geomNode.geometry = geom;
}

382 // protected JGeometry geometry;
rep+ protected Geometry geometry;

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.XlinkWorldFile

134 // JGeometry geom = new JGeometry(content.get(4), content.get(5), dbSrid);
// STRUCT obj = JGeometry.store(geom, batchConn);
Point ptGeom = new Point(content.get(4), content.get(5));
Geometry geom = PGgeometry.geomFromString(
    "SRID=" + dbSrid + ";" + ptGeom);
PGgeometry pgGeom = new PGgeometry(geom);

```

## 5.2 From 3DCityDB back to CityGML

Simply said, the export works the other way around. In *Oracle* the `ResultSet` is casted into the `STRUCT` data type and then "loaded" into a `JGeometry` object. The *PostGIS* way works in a similar manner with `PGgeometry.getGeometry`. In *Oracle* `JGeometry` can easily be transferred to arrays and processed back again into list-elements for the CityGML primitives. The `ELEM_INFO_ARRAY` helps to distinguish between geometric types. The *PostGIS* JDBC offers different sub-classes of `Geometry.java`. `ComposedGeom` and `MultiLineString` were used for addressing subgeometries. Fortunately this did not lead to conflicts against the names of the `citygml4j.lib`.

de.tub.citydb.modules.citygml.exporter.database.content.**DBAppearance**

```

822 // STRUCT struct = (STRUCT)rs.getObject("GT_REFERENCE_POINT");
rep+ // if (!rs.wasNull() && struct != null) {
//     JGeometry jGeom = JGeometry.load(struct);
//     double[] point = jGeom.getPoint();
//
//     if (point != null && point.length >= 2) {
//         Point referencePoint = new PointImpl();
//         List<Double> value = new ArrayList<Double>();
//         value.add(point[0]);
//         value.add(point[1]);
//     }
//     PGGeometry pgGeom = (PGGeometry)rs.getObject("GT_REFERENCE_POINT");
//     if (!rs.wasNull() && pgGeom != null) {
//         Geometry geom = pgGeom.getGeometry();
//         Point referencePoint = new PointImpl();
//         List<Double> value = new ArrayList<Double>();
//         value.add(geom.getPoint(0).getX());
//         value.add(geom.getPoint(0).getY());

```

de.tub.citydb.modules.citygml.exporter.database.content.**DBCityObject**

```

164 // double[] points = geom.getMBR();

170 // if (geom.getDimension() == 2) {
//     lower = new Point(points[0], points[1], 0);
//     upper = new Point(points[2], points[3], 0);
// } else {
//     lower = new Point(points[0], points[1], points[2]);
//     upper = new Point(points[3], points[4], points[5]);
// }
// if (geom.getDimension() == 2) {
//     lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y, 0);
//     upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y, 0);
// } else {
//     lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
//         geom.getFirstPoint().z);
//     upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
//         geom.getPoint(2).z);

```

de.tub.citydb.modules.citygml.exporter.database.content.**DBGeneralization**

```

121 // double[] points = geom.getOrdinatesArray();
// Point lower = new Point(points[0], points[1], points[2]);
// Point upper = new Point(points[3], points[4], points[5]);
// Point lower = new Point(geom.getFirstPoint().x, geom.getFirstPoint().y,
//     geom.getFirstPoint().z);
// Point upper = new Point(geom.getPoint(2).x, geom.getPoint(2).y,
//     geom.getPoint(2).z);

```

de.tub.citydb.modules.citygml.exporter.database.content.**DBStGeometry**

```

94  // public PointProperty getPoint(JGeometry geom, boolean setSrsName) {
//      PointProperty pointProperty = null;
//      if (geom != null && geom.getType() == JGeometry.GTYPE_POINT) {
//          pointProperty = new PointPropertyImpl();
//          int dimensions = geom.getDimensions();
//
//          double[] pointCoord = geom.getPoint();
//
//          if (pointCoord != null && pointCoord.length >= dimensions) {
//              Point point = new PointImpl();
//
//              List<Double> value = new ArrayList<Double>();
//              for (int i = 0; i < dimensions; i++)
//                  value.add(pointCoord[i]);
//          }
//          public PointProperty getPoint(Geometry geom, boolean setSrsName) {
//              PointProperty pointProperty = null;
//
//              if (geom != null && geom.getType() == 1) {
//                  pointProperty = new PointPropertyImpl();
//                  int dimensions = geom.getDimension();
//
//                  if (dimensions == 2) {
//                      Point point = new PointImpl();
//
//                      List<Double> value = new ArrayList<Double>();
//                      value.add(geom.getPoint(0).getX());
//                      value.add(geom.getPoint(0).getY());
//                      .
//                      .
//                  }
//                  if (dimensions == 3) {
//
//                      Point point = new PointImpl();
//                      List<Double> value = new ArrayList<Double>();
//                      value.add(geom.getPoint(0).getX());
//                      value.add(geom.getPoint(0).getY());
//                      value.add(geom.getPoint(0).getZ());
//                  }
//              }
//          }
//      }
//      public PolygonProperty getPolygon(JGeometry geom, boolean setSrsName) {
//          PolygonProperty polygonProperty = null;
//          if (geom != null && geom.getType() == JGeometry.GTYPE_POLYGON) {
//              polygonProperty = new PolygonPropertyImpl();
//              Polygon polygon = new PolygonImpl();
//              int dimensions = geom.getDimensions();
//
//              int[] elemInfoArray = geom.getElemInfo();
//              double[] ordinatesArray = geom.getOrdinatesArray();
//
//              if (elemInfoArray.length < 3 || ordinatesArray.length == 0)
//                  return null;
//
//              List<Integer> ringLimits = new ArrayList<Integer>();
//              for (int i = 3; i < elemInfoArray.length; i += 3)
//                  ringLimits.add(elemInfoArray[i] - 1);
//          }
//      }

```

```

//          ringLimits.add(ordinatesArray.length);
//
//          boolean isExterior = elemInfoArray[1] == 1003;
//          int ringElem = 0;
//          for (Integer curveLimit : ringLimits) {
//              List<Double> values = new ArrayList<Double>();
//
//              for ( ; ringElem < curveLimit; ringElem++)
//                  values.add(ordinatesArray[ringElem]);
//
//              if (isExterior) {
public PolygonProperty getPolygon(Geometry geom, boolean setSrsName) {
    PolygonProperty polygonProperty = null;

    if (geom != null && geom.getType() == 3) {
        polygonProperty = new PolygonPropertyImpl();
        Polygon polygon = new PolygonImpl();
        int dimensions = geom.getDimension();

        if (geom.getValue() == null)
            return null;

        org.postgis.Polygon polyGeom = (org.postgis.Polygon) geom;

        for (int i = 0; i < polyGeom.numRings(); i++){
            List<Double> values = new ArrayList<Double>();

            if (dimensions == 2)
                for (int j=0; j<polyGeom.getRing(i).numPoints(); j++){
                    values.add(polyGeom.getRing(i).getPoint(j).x);
                    values.add(polyGeom.getRing(i).getPoint(j).y);
                }

            if (dimensions == 3)
                for (int j=0; j<polyGeom.getRing (i).numPoints(); j++){
                    values.add(polyGeom.getRing (i).getPoint(j).x);

                    values.add(polyGeom.getRing (i).getPoint(j).y);
                    values.add(polyGeom.getRing (i).getPoint(j).z);
                }
            //isExterior
            if (i == 0) {
208 // public MultiPointProperty getMultiPointProperty(JGeometry geom, boolean
rep // setSrsName) {
//     MultiPointProperty multiPointProperty = null;
//
//     if (geom != null) {
//         multiPointProperty = new MultiPointPropertyImpl();
//         MultiPoint multiPoint = new MultiPointImpl();
//         int dimensions = geom.getDimensions();
//
//         if (geom.getType() == JGeometry.GTYPE_MULTIPOINT) {
//             double[] ordinates = geom.getOrdinatesArray();
//
//             for (int i = 0; i < ordinates.length; i += dimensions) {
//                 Point point = new PointImpl();

```

```

//
//      List<Double> value = new ArrayList<Double>();
//
//      for (int j = 0; j < dimensions; j++)
//          value.add(ordinates[i + j]);
//
//          .
//          .
//      }
//  } else if (geom.getType() == JGeometry.GTYPE_POINT) {
//      ..
public MultiPointProperty getMultiPointProperty(Geometry geom, boolean
setSrsName) {
    MultiPointProperty multiPointProperty = null;

    if (geom != null) {
        multiPointProperty = new MultiPointPropertyImpl();
        MultiPoint multiPoint = new MultiPointImpl();
        int dimensions = geom.getDimension();

        if (geom.getType() == 4) {
            List<Double> value = new ArrayList<Double>();
            Point point = new PointImpl();

            if (dimensions == 2)
                for (int i = 0; i < geom.numPoints(); i++) {
                    value.add(geom.getPoint(i).x);
                    value.add(geom.getPoint(i).y);
                }
            if (dimensions == 3)
                for (int i = 0; i < geom.numPoints(); i++) {
                    value.add(geom.getPoint(i).x);
                    value.add(geom.getPoint(i).y);
                    value.add(geom.getPoint(i).z);
                }
            .
            .
        }
    }
    else if (geom.getType() == 1) {
        Point point = new PointImpl();

        List<Double> value = new ArrayList<Double>();
        value.add(geom.getPoint(0).x);
        value.add(geom.getPoint(0).y);

        if (dimensions == 3)
            value.add(geom.getPoint(0).z);
    }
}
355 // public MultiCurveProperty getMultiCurveProperty(JGeometry geom, boolean
rep // setSrsName) {
//     MultiCurveProperty multiCurveProperty = null;
//
//     if (geom != null) {
//         multiCurveProperty = new MultiCurvePropertyImpl();
//         MultiCurve multiCurve = new MultiCurveImpl();
//         int dimensions = geom.getDimensions();
//

```



```

//          if (geom.getType() == JGeometry.GTYPE_MULTICURVE ) {
//              int[] elemInfoArray = geom.getElemInfo();
//              double[] ordinatesArray = geom.getOrdinatesArray();
//
//              if (elemInfoArray.length < 3 ||
//                  ordinatesArray.length == 0)
//                  return null;
//
//              List<Integer> curveLimits = new ArrayList<Integer>();
//              for (int i = 3; i < elemInfoArray.length; i += 3)
//                  curveLimits.add(elemInfoArray[i] - 1);
//
//              curveLimits.add(ordinatesArray.length);
//
//              int curveElem = 0;
//              for (Integer curveLimit : curveLimits) {
//                  List<Double> values = new ArrayList<Double>();
//
//                  for ( ; curveElem < curveLimit; curveElem++)
//                      values.add(ordinatesArray[curveElem]);
//                  .
//                  .
//                  curveElem = curveLimit;
//              }
//          }
//      }
//      else if (geom.getType() == JGeometry.GTYPE_CURVE ) {
//          double[] ordinatesArray = geom.getOrdinatesArray();
//          List<Double> value = new ArrayList<Double>();
//
//          for (int i = 0; i < ordinatesArray.length; i++)
//              value.add(ordinatesArray[i]);
//      }
public MultiCurveProperty getMultiCurveProperty(Geometry geom, boolean
setSrsName) {
    MultiCurveProperty multiCurveProperty = null;

    if (geom != null) {
        multiCurveProperty = new MultiCurvePropertyImpl();
        MultiCurve multiCurve = new MultiCurveImpl();
        int dimensions = geom.getDimension();

        if (geom.getType() == 5) {
            MultiLineString mlineGeom = (MultiLineString)geom;

            for (int i = 0; i < mlineGeom.numLines(); i++){

                List<Double> values = new ArrayList<Double>();

                if (dimensions == 2)
                    for (int j=0; j<mlineGeom.getLine(i).numPoints();
                        j++){
                        values.add(mlineGeom.getLine(i).getPoint(j).x);
                        values.add(mlineGeom.getLine(i).getPoint(j).y);
                    }
                if (dimensions == 3)
                    for (int j=0; j<mlineGeom.getLine(i).numPoints();
                        j++){
                        values.add(mlineGeom.getLine(i).getPoint(j).x);

```

```

        values.add(mlineGeom.getLine(i).getPoint(j).y);
        values.add(mlineGeom.getLine(i).getPoint(j).z);
    }
    .
    .
}
}
else if (geom.getType() == 2) {
    List<Double> value = new ArrayList<Double>();

    if (dimensions == 2)
        for (int i = 0; i < geom.numPoints(); i++) {
            value.add(geom.getPoint(i).x);
            value.add(geom.getPoint(i).y);
        }
    if (dimensions == 3)
        for (int i = 0; i < geom.numPoints(); i++) {
            value.add(geom.getPoint(i).x);
            value.add(geom.getPoint(i).y);
            value.add(geom.getPoint(i).z);
        }
}

```

de.tub.citydb.util.database.**DBUtil**

```

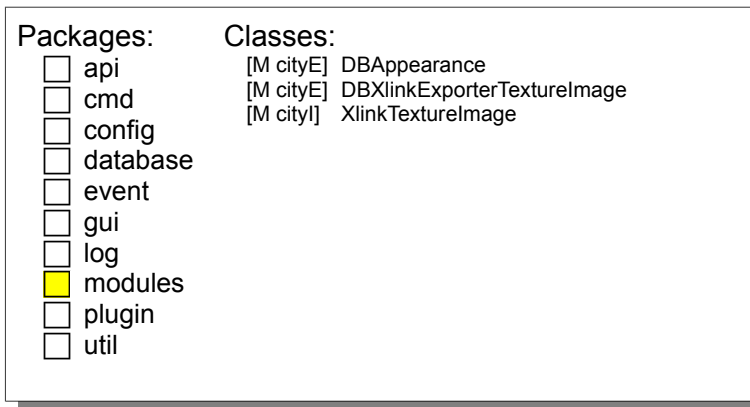
308 // STRUCT struct = (STRUCT)rs.getObject(1);
rep+ // if (!rs.wasNull() && struct != null) {
//     JGeometry jGeom = JGeometry.load(struct);
//     int dim = jGeom.getDimensions();
PGGeometry pgGeom = (PGGeometry)rs.getObject(1);
if (!rs.wasNull() && pgGeom != null) {
    Geometry geom = pgGeom.getGeometry();
    int dim = geom.getDimension();
}

```

## 5.3 Synchronization of geometric functions

It is proven that JGeometry's method `store(JGeometry)` is not threadsafe and deadlocks can occur. This problem is avoided by synchronizing the storing of JGeometries into STRUCT objects with a Java-Reentrant-Lock (inside `SyncJGeometry.java`). Until now no such problem occurred for the *PostGIS* version.

## 6. How to deal with textures



As the `ORDImage` data type differs a lot from the `BYTEA` data type in *PostgreSQL* it is not surprising that the im- and export of textures had to be changed in many aspects. With `ORDImage` it is possible to query metadata from the images and also use functions similar to a graphic processing software. Some of these features are called in the `DBAppearance` class (see also chapter 3.3). Overall, the *3DCityDB* hardly uses the abilities of `ORDImage`. Even Oracle itself recommended the use of `BLOBs` for the *3DCityDB* to the developers.

### 6.1 Import of textures

As seen on the following examples the code for importing textures could be reduced to a few lines. Inserting `ORDImages` works as follows:

1. initialization in the database with `ordimage.init()`
2. a "select for update" locks the `ResultSet`-cursor for the row to be updated
3. the database-`ORDImage` is transferred to a java-`ORDImage` but still empty
4. `loadDataFromInputStream` fills the empty `ORDImage.java`
5. `setORADATA` sets the `ORDImage.java` in the `PreparedStatement` which inserts the data by updating the table `Surface_Data`

With `BLOBs` the output of the `InputStream` can directly be set in the `PreparedStatement` with `setBinaryStream`.

de.tub.citydb.modules.citygml.importer.database.xlink.resolver.**XlinkTextureImage**

```
77 // psPrepare = externalFileConn.prepareStatement(
    "update SURFACE_DATA set TEX_IMAGE=ordimage.init() where ID=?");
// psSelect = externalFileConn.prepareStatement(
    "select TEX_IMAGE from SURFACE_DATA where ID=? for update");
// psInsert = (OraclePreparedStatement)externalFileConn.prepareStatement(
    "update SURFACE_DATA set TEX_IMAGE=? where ID=?");
psInsert = externalFileConn.prepareStatement(
    "update SURFACE_DATA set TEX_IMAGE=? where ID=?");
```

```
113+ // // second step: prepare ORDIMAGE
// psPrepare.setLong(1, xlink.getId());
// psPrepare.executeUpdate();
//
// // third step: get prepared ORDIMAGE to fill it with contents
// psSelect.setLong(1, xlink.getId());
// OracleResultSet rs = (OracleResultSet)psSelect.executeQuery();
// if (!rs.next()) {
//     LOG.error("Database error while importing texture file '" +
//         imageFileName + "'.");
//
//     rs.close();
//
//     externalFileConn.rollback();
//     return false;
// }

120 // OrdImage imgProxy = (OrdImage)rs.getORADData(
//     1, OrdImage.getORADDataFactory());
// rs.close();
// boolean letDBdetermineProperties = true;
// if (isRemote) {
//     InputStream stream = imageURL.openStream();
//     imgProxy.loadDataFromInputStream(stream);
// } else {
//     imgProxy.loadDataFromFile(imageFileName);
//
//     // determing image formats by file extension
//     int index = imageFileName.lastIndexOf('.');
//     if (index != -1) {
//         String extension = imageFileName.substring(
//             index + 1, imageFileName.length());
//
//         if (extension.toUpperCase().equals("RGB")) {
//             imgProxy.setMimeType("image/rgb");
//             imgProxy.setFormat("RGB");
//             imgProxy.setContentLength(1);
//
//             letDBdetermineProperties = false;
//         }
//     }
// }
// if (letDBdetermineProperties)
//     imgProxy.setProperties();
//
// psInsert.setORADData(1, imgProxy);
// psInsert.setLong(2, xlink.getId());
// psInsert.execute();
// imgProxy.close();
InputStream in = null;
if (isRemote) {
    in = imageURL.openStream();
} else {
    in = new FileInputStream(imageFile);
}
psInsert.setBinaryStream(1, in, in.available());
```

```
psInsert.setLong(2, xlink.getId());
psInsert.execute();

in.close()
externalFileConn.commit();
return true;
```

## 6.2 Export of textures

The export of textures in the *Oracle* version only needs a few lines but is also very *ORDImage*-specific. Two ways exist for the *PostgreSQL*'s *BYTEA* data type. No performance differences could be noticed until now. The second option looks more efficient as the reading of the images is programmed in a stream-based way. Due to information found in mailing list archives *BYTEAs* cannot be read stream-based. They will always be loaded completely into the RAM. This would cause problems when dealing with bigger texture atlases. An alternative is offered by using large objects (LOBs). LOBs are stored separately from the *PostgreSQL* database and are referenced by an object identifier (OID).

The im- and export of LOBs was implemented as well and works as good as the *BYTEA* solution. The user has to pay attention when deleting table entries with the *OID* data type. Deleting the reference won't delete the texture file. This has to be done with certain functions (*vacuumlo*, *lo\_unlink*). For this release the code for im- and exporting LOBs is not used and therefore not shown here. It is up to the user to change the source code and the corresponding columns (*BYTEA* → *OID*) in the SQL scripts in order to test it (see below).

Java classes:

```
de.tub.citydb.modules.citygml.importer.database.xlink.XlinkTextureImage
de.tub.citydb.modules.citygml.importer.database.xlink.XlinkLibraryObject
de.tub.citydb.modules.citygml.exporter.database.xlink.DBXlinkExporterTextureImage
de.tub.citydb.modules.citygml.exporter.database.xlink.DBXlinkExporterLibraryObject
```

SQL:

```
table Surface_Data, column tex_image
table Implicit_Geometry, column library_object
```

```
de.tub.citydb.modules.citygml.exporter.database.xlink.DBXlinkExporterTextureImage
```

```
128 // OracleResultSet rs = (OracleResultSet)psTextureImage.executeQuery();
    ResultSet rs = (ResultSet)psTextureImage.executeQuery();

143 // // read oracle image data type
    // OrdImage imgProxy = (OrdImage)rs.getORADData(
    //     1, OrdImage.getORADDataFactory());
    // rs.close();
    //
    // if (imgProxy == null) {
```

```
// LOG.error("Database error while reading texture file: " + fileName);
// return false;
// }
//
// try {
//     imgProxy.getDataInFile(fileURI);
// } catch (IOException ioEx) {
//     LOG.error("Failed to write texture file " + fileName + ": " +
//         ioEx.getMessage());
//     return false;
// } finally {
//     imgProxy.close();
// }
```

1st way:

```
byte[] imgBytes = rs.getBytes(1);
try {
    FileOutputStream fos = new FileOutputStream(fileURI);
    fos.write(imgBytes);
    fos.close();
} catch (FileNotFoundException fnfEx) {
    LOG.error("File not found " + fileName + ": " + fnfEx.getMessage());
} catch (IOException ioEx) {
    LOG.error("Failed to write texture file " + fileName + ": " +
        ioEx.getMessage());
    return false;
}
```

2nd way:

```
InputStream imageStream = rs.getBinaryStream(1);
if (imageStream == null) {
    LOG.error("Database error while reading texture file: " + fileName);
    return false;
}
try {
    byte[] imgBuffer = new byte[1024];
    FileOutputStream fos = new FileOutputStream(fileURI);
    int l;
    while ((l = imageStream.read(imgBuffer)) > 0) {
        fos.write(imgBuffer, 0, l);
    }
    fos.close();
} catch (FileNotFoundException fnfEx) {
    LOG.error("File not found " + fileName + ": " + fnfEx.getMessage());
} catch (IOException ioEx) {
    LOG.error("Failed to write texture file " + fileName + ": " +
        ioEx.getMessage());
    return false; }
}
```

## 7. The batchsize of PostgreSQL

Packages:	Classes:
<input type="checkbox"/> api	[C] Internal
<input type="checkbox"/> cmd	[C] UpdateBatching
<input checked="" type="checkbox"/> config	[M cityE] DBExportCache
<input type="checkbox"/> database	[M city!] DBImportXlinkResolverWorker
<input type="checkbox"/> event	[M city!] DBImportXlinkWorker
<input type="checkbox"/> gui	[M city!] DBAddress
<input type="checkbox"/> log	[M city!] DBAddressToBuilding
<input checked="" type="checkbox"/> modules	[M city!] DBAppearance
<input type="checkbox"/> plugin	[M city!] DBAppearToSurfaceData
<input type="checkbox"/> util	[M city!] DBBuilding
	[M city!] DBBuildingFurniture
	[M city!] DBBuildingInstallation
	[M city!] DBCityFurniture
	[M city!] DBCityObject
	[M city!] DBCityObjectGenericCityObject
	[M city!] DBCityObjectGroup
	[M city!] DBExternalReference
	[M city!] DBGenericCityObject
	[M city!] DBImplicitGeometry
	[M city!] DBLandUse
	[M city!] DBOpening
	[M city!] DBOpeningToThemSurface
	[M city!] DBPlantCover
	[M city!] DBReliefComponent
	[M city!] DBReliefFeatToRelComp
	[M city!] DBReliefFeature
	[M city!] DBRoom
	[M city!] DBSolitaryVegetatObject
	[M city!] DBSurfaceData
	[M city!] DBSurfaceGeometry
	[M city!] DBThematicSurface
	[M city!] DBTrafficArea
	[M city!] DBTransportationComplex
	[M city!] DBWaterBodyToWaterBndSrf
	[M city!] DBWaterBody
	[M city!] DBWaterBoundarySurface
	[M city!] DBImportCache
	[M city!] DBXlinkImporterBasic
	[M city!] DBXlinkImporterDeprecatedMaterial
	[M city!] DBXlinkImporterGroupToCityObject
	[M city!] DBXlinkImporterLibraryObject
	[M city!] DBXlinkImporterLinearRing
	[M city!] DBXlinkImporterSurfacegeometry
	[M city!] DBXlinkImporterTextureAssociation
	[M city!] DBXlinkImporterTextureFile
	[M city!] DBXlinkImporterTextureParam
	[M city!] XlinkBasic
	[M city!] XlinkDeprecatedMaterial
	[M city!] XlinkGroupToCityObject
	[M city!] XlinkSurfaceGeometry
	[M city!] XlinkTexCoordList
	[M city!] XlinkTextureAssociation
	[M city!] XlinkTextureParam
	[M city!] XlinkWorldFile
	[M city!] ResourcesPanel

The maximum batchsize of *PostgreSQL* was set to 10000. More might be possible but was not tested. This change in the `Internal` class caused several classes to be changed for compiling. They are all listed in the overview-box.

de.tub.citydb.config.internal.**Internal**

```
40    //    public static final int ORACLE_MAX_BATCH_SIZE = 65535;
    public static final int POSTGRES_MAX_BATCH_SIZE = 10000;
```

In the following classes no equivalent methods could be found for the Java PreparedStatement. The psDrain-batch is now executed and not sent.

de.tub.citydb.modules.citygml.exporter.database.gmlid.**DBExportCache**

de.tub.citydb.modules.citygml.importer.database.gmlid.**DBImportCache**

```
84    // ((OraclePreparedStatement)psDrains[i]).setExecuteBatch(batchSize);

145   // ((OraclePreparedStatement)psDrain).sendBatch();
    psDrain.executeBatch();
```

## 8. Workspace Management

Packages:	Classes:
<input checked="" type="checkbox"/> api	[A] DatabaseController
<input type="checkbox"/> cmd	[C] Internal
<input type="checkbox"/> config	[C] Database
<input checked="" type="checkbox"/> database	[C] Workspace
<input type="checkbox"/> event	[C] Workspaces
<input type="checkbox"/> gui	[D] DatabaseConnectionPool
<input type="checkbox"/> log	[D] DatabaseControllerImpl
<input checked="" type="checkbox"/> modules	[M cityE] DBExportCache
<input type="checkbox"/> plugin	[M cityE] DBExportXlinkWorker
<input checked="" type="checkbox"/> util	[M cityE] DBExporter
	[M cityE] DBSplitter
	[M cityE] ExportPanel
	[M cityI] DBImportCache
	[M cityI] DBImportXlinkResolverWorker
	[M cityI] DBImporter
	[M cityI] ImportPanel
	[M DB] BoundingBoxOperation
	[M DB] DatabaseOperationsPanel
	[M DB] ReportOperation
	[U] DBUtil
	[U] Util

*PostgreSQL* does not offer a workspace or history management like *Oracle* does. Every part in the Java code concerning these workspace features was uncommented but not deleted as there might be a solution for database versioning in the future. The affected packages are colored orange.



## 9. KML-Exporter

Packages:	Classes:
<input type="checkbox"/> api	[M kml] KmlExportWorker
<input type="checkbox"/> cmd	[M kml] KmlExporter
<input type="checkbox"/> config	[M kml] BalloonTemplateHandlerImpl
<input type="checkbox"/> database	[M kml] CityObjectGroup
<input type="checkbox"/> event	[M kml] ColladaBundle
<input type="checkbox"/> gui	[M kml] KmlExporterManager
<input type="checkbox"/> log	[M kml] KmlGenericObject
<input type="checkbox"/> modules	[M kml] KmlSplitter
<input type="checkbox"/> plugin	[M kml] Queries
<input type="checkbox"/> util	

Due to the modular architecture of the *Importer/Exporter* the port of the *KML-Exporter* only affected classes of the KML module. The code design differs from the CityGML module. Database queries are collected in one central class and were used as string constants in other classes. Database geometries were parsed into an array to create the KML primitives. Until now it is only possible to export buildings. In the future a generic class will be used as a parent for subclasses of other thematic modules of the *3DCityDB*.

### 9.1 Queries

de.tub.citydb.modules.kml.database.**Queries**

```

53  // public static final String GET_GMLIDS =
rep  // "SELECT co.gmlid, co.class_id " +
    // "FROM CITYOBJECT co " +
    // "WHERE " +
    // "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2002, ?, null, " +
    // "MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1), " +
    // "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), " +
    // "'mask=overlapbdydisjoint') ='TRUE') " +
    // "UNION ALL " +
    // "SELECT co.gmlid, co.class_id " +
    // "FROM CITYOBJECT co " +
    // "WHERE " +
    // "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2003, ?, null,
    // "MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), " +
    // "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), " +
    // "'mask=inside+coveredby') ='TRUE') " +
    // "UNION ALL " +
    // "SELECT co.gmlid, co.class_id " +
    // "FROM CITYOBJECT co " +
    // "WHERE " +
    // "(SDO_RELATE(co.envelope, MDSYS.SDO_GEOMETRY(2003, ?, null, " +
    // "MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), " +
    // "MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)), 'mask=equal') ='TRUE') "
    // + "ORDER BY 2"; // ORDER BY co.class_id

```

```

public static final String GET_GMLIDS =
    "SELECT co.gmlid, co.class_id " +
    "FROM CITYOBJECT co " +
    "WHERE " +
    // overlap
    "ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*T***T**') ='TRUE' "
    + "UNION ALL " +
    "SELECT co.gmlid, co.class_id " +
    "FROM CITYOBJECT co " +
    "WHERE " +
    "    (ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*F**F***')
    = 'TRUE' OR " + // inside and coveredby
    "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '*TF**F***')
    = 'TRUE' OR " + // coveredby
    "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '**FT*F***')
    = 'TRUE' OR " + // coveredby
    "ST_Relate(co.envelope, ST_GeomFromEWKT(?), '**F*TF***')
    = 'TRUE') " + // coveredby
    "UNION ALL " +
    "SELECT co.gmlid, co.class_id " +
    "FROM CITYOBJECT co " +
    "WHERE " +
    "    ST_Relate(co.envelope, ST_GeomFromEWKT(?), 'T*F***FFF')
    = 'TRUE' " + // equal
    "ORDER BY 2"; // ORDER BY co.class_id*/

100 // public static final String QUERY_EXTRUDED_HEIGHTS =
    // "SELECT " + // "b.measured_height, " +
    // "SDO_GEOM.SDO_MAX_MBR_ORDINATE(co.envelope, 3) -
    //     SDO_GEOM.SDO_MIN_MBR_ORDINATE(co.envelope, 3) AS
    //     envelope_measured_height " +
    // "FROM CITYOBJECT co " + // ", BUILDING b " +
    // "WHERE " +
    // "    co.gmlid = ?"; // + " AND b.building_root_id = co.id";
public static final String GET_EXTRUDED_HEIGHT =
    "SELECT " + // "b.measured_height, " +
    "ST_ZMax(Box3D(co.envelope)) - ST_ZMin(Box3D(co.envelope)) AS
    envelope_measured_height " +
    "FROM CITYOBJECT co " + // ", BUILDING b " +
    "WHERE co.gmlid = ?"; // + " AND b.building_root_id = co.id";

114 // public static final String INSERT_GE_ZOFFSET =
    // "INSERT INTO CITYOBJECT_GENERICATTRIB (ID, ATTRNAME, DATATYPE,
    //     STRVAL, CITYOBJECT_ID) " +
    // "VALUES (CITYOBJECT_GENERICATT_SEQ.NEXTVAL, ?, 1, ?,
    //     (SELECT ID FROM CITYOBJECT WHERE gmlid = ?))";
    //
    // public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
    //     "SELECT SDO_CS.TRANSFORM(?, 4326) FROM DUAL";
    //
    // public static final String TRANSFORM_GEOMETRY_TO_WGS84_3D =
    //     "SELECT SDO_CS.TRANSFORM(?, 4329) FROM DUAL";
    //
    // public static final String GET_ENVELOPE_IN_WGS84_FROM_GML_ID =
    //     "SELECT SDO_CS.TRANSFORM(co.envelope, 4326) " +
    //     "FROM CITYOBJECT co " +
    //     "WHERE co.gmlid = ?";

```

```

//
//      public static final String GET_ENVELOPE_IN_WGS84_3D_FROM_GML_ID =
//          "SELECT SDO_CS.TRANSFORM(co.envelope, 4329) " +
//          "FROM CITYOBJECT co " +
//          "WHERE co.gmlid = ?";
public static final String INSERT_GE_ZOFFSET =
    "INSERT INTO CITYOBJECT_GENERICATTRIB (ID, ATTRNAME, DATATYPE, " +
    "STRVAL, CITYOBJECT_ID) " +
    "VALUES (nextval('CITYOBJECT_GENERICATTRIB_ID_SEQ'), ?, 1, ?, " +
    "(SELECT ID FROM CITYOBJECT WHERE gmlid = ?))";

public static final String TRANSFORM_GEOMETRY_TO_WGS84 =
    "SELECT ST_Transform(?, 4326)";

public static final String TRANSFORM_GEOMETRY_TO_WGS84_3D =
    "SELECT ST_Transform(?, 94329)";

public static final String GET_ENVELOPE_IN_WGS84_FROM_GML_ID =
    "SELECT ST_Transform(co.envelope, 4326) " +
    "FROM CITYOBJECT co " +
    "WHERE co.gmlid = ?";

public static final String GET_ENVELOPE_IN_WGS84_3D_FROM_GML_ID =
    "SELECT ST_Transform(co.envelope, 94329) " +
    "FROM CITYOBJECT co " +
    "WHERE co.gmlid = ?";

```

The following example is a bit tricky. In *Oracle* it is possible to do a sort of pyramid aggregation. That means aggregations are primarily done on smaller groups which are then aggregated to bigger groups and so on (see GROUP BY clauses at the end of the query). Depending on the size of the `surface_geometry` table it will work much faster than the *PostGIS* ST\_Union operation.

```

591 //      public static final String
rep //      BUILDING GETAggregateGeometriesForLod2OrHigher =
//          "SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
//              <TOLERANCE>)) aggr_geom " +
//          "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
//              <TOLERANCE>)) aggr_geom " +
//          "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,
//              <TOLERANCE>)) aggr_geom " +
//          "FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(simple_geom,
//              <TOLERANCE>)) aggr_geom " +
//          "FROM (" +
//          "
//          "SELECT * FROM (" +
//          "SELECT * FROM (" +
//          "
//          "SELECT geodb_util.to_2d(sg.geometry, <2D_SRID>) AS
//              simple_geom " +
//          //"SELECT geodb_util.to_2d(sg.geometry, (select srid from
//              database_srs)) AS simple_geom " +
//          //"SELECT sg.geometry AS simple_geom " +
//          "FROM SURFACE_GEOMETRY sg " +
//          "WHERE " +

```

```

//          "sg.root_id IN( " +
//          "SELECT b.lod<LoD>_geometry_id " +
//          "FROM CITYOBJECT co, BUILDING b " +
//          "WHERE "+
//          "co.gmlid = ? " +
//          "AND b.building_root_id = co.id " +
//          "AND b.lod<LoD>_geometry_id IS NOT NULL " +
//          "UNION " +
//          "SELECT ts.lod<LoD>_multi_surface_id " +
//          "FROM CITYOBJECT co, BUILDING b, THEMATIC_SURFACE ts " +
//          "WHERE "+
//          "co.gmlid = ? " +
//          "AND b.building_root_id = co.id " +
//          "AND ts.building_id = b.id " +
//          "AND ts.lod<LoD>_multi_surface_id IS NOT NULL "+
//          ") " +
//          "AND sg.geometry IS NOT NULL" +
//
//          ") WHERE sdo_geom.validate_geometry(simple_geom, <TOLERANCE>)
//          = 'TRUE'" +
//          ") WHERE sdo_geom.sdo_area(simple_geom, <TOLERANCE>) >
//          <TOLERANCE>" +
//
//          ") " +
//          "GROUP BY mod(rownum, <GROUP_BY_1>) " +
//          ") " +
//          "GROUP BY mod (rownum, <GROUP_BY_2>) " +
//          ") " +
//          "GROUP BY mod (rownum, <GROUP_BY_3>) " +
//          ")";
"SELECT ST_Union(get_valid_area.simple_geom) " +
"FROM (" +
"SELECT * FROM (" +
"SELECT * FROM (" +
"SELECT ST_Force_2D(sg.geometry) AS simple_geom " +
"FROM SURFACE_GEOMETRY sg " +
"WHERE " +
"sg.root_id IN( " +
"SELECT b.lod<LoD>_geometry_id " +
"FROM CITYOBJECT co, BUILDING b " +
"WHERE "+
"co.gmlid = ? " +
"AND b.building_root_id = co.id " +
"AND b.lod<LoD>_geometry_id IS NOT NULL " +
"UNION " +
"SELECT ts.lod<LoD>_multi_surface_id " +
"FROM CITYOBJECT co, BUILDING b, THEMATIC_SURFACE ts " +
"WHERE "+
"co.gmlid = ? " +
"AND b.building_root_id = co.id " +
"AND ts.building_id = b.id " +
"AND ts.lod<LoD>_multi_surface_id IS NOT NULL "+
") " +
"AND sg.geometry IS NOT NULL) AS get_geoms " +
"WHERE ST_IsValid(get_geoms.simple_geom) = 'TRUE') AS get_valid_geoms "
"WHERE ST_Area(get_valid_geoms.simple_geom) > <TOLERANCE>) AS
get_valid_area"; // PostgreSQL-Compiler needs subquery-aliases

```

Like for the CityGML export the RELATE operations can only be sent as an PreparedStatement to the *PostGIS* database when using the whole WKT-string as the bind variable.

de.tub.citydb.modules.kml.database.**KmlSplitter**

```

264 //      BoundingBox tile =
rep //      exportFilter.getBoundingBoxFilter().getFilterState();
//      OracleResultSet rs = null;
//      PreparedStatement spatialQuery = null;
//      try {
//          spatialQuery =
//          connection.prepareStatement(TileQueries.QUERY_GET_GMLIDS);
//          int srid =
//          DatabaseConnectionPool.getInstance().
//          getActiveConnectionMetaData().getReferenceSystem().getSrid();
//          spatialQuery.setInt(1, srid);
//          // coordinates for inside
//          spatialQuery.setDouble(2, tile.getLowerLeftCorner().getX());
//          spatialQuery.setDouble(3, tile.getLowerLeftCorner().getY());
//          spatialQuery.setDouble(4, tile.getUpperRightCorner().getX());
//          spatialQuery.setDouble(5, tile.getUpperRightCorner().getY());
//          spatialQuery.setInt(6, srid);
//          // coordinates for overlapbdydisjoint
//          spatialQuery.setDouble(7, tile.getLowerLeftCorner().getX());
//          spatialQuery.setDouble(8, tile.getUpperRightCorner().getY());
//          spatialQuery.setDouble(9, tile.getLowerLeftCorner().getX());
//          spatialQuery.setDouble(10, tile.getLowerLeftCorner().getY());
//          spatialQuery.setDouble(11, tile.getUpperRightCorner().getX());
//          spatialQuery.setDouble(12, tile.getLowerLeftCorner().getY());
//          rs = (OracleResultSet)query.executeQuery();
ResultSet rs = null;
PreparedStatement query = null;
String lineGeom = null;
String polyGeom = null;

try {
    if (filterConfig.isSetComplexFilter() &&
        filterConfig.getComplexFilter().getTiledBoundingBox().isSet()) {

        query =connection.prepareStatement(
            Queries.CITYOBJECTGROUP_MEMBERS_IN_BBOX);

        BoundingBox tile = exportFilter.getBoundingBoxFilter()
                                .getFilterState();

        int srid = dbSrs.getSrid();

        lineGeom = "SRID=" + srid + ";LINESTRING(" +
            tile.getLowerLeftCorner().getX() + " " +
            tile.getUpperRightCorner().getY() + "," +
            tile.getLowerLeftCorner().getX() + " " +
            tile.getLowerLeftCorner().getY() + "," +
            tile.getUpperRightCorner().getX() + " " +

```

```

        tile.getLowerLeftCorner().getY() + ") '";

        polyGeom = "SRID=" + srid + ";POLYGON((" +
            tile.getLowerLeftCorner().getX() + " " +
            tile.getLowerLeftCorner().getY() + "," +
            tile.getLowerLeftCorner().getX() + " " +
            tile.getUpperRightCorner().getY() + "," +
            tile.getUpperRightCorner().getX() + " " +
            tile.getUpperRightCorner().getY() + "," +
            tile.getUpperRightCorner().getX() + " " +
            tile.getLowerLeftCorner().getY() + "," +
            tile.getLowerLeftCorner().getX() + " " +
            tile.getLowerLeftCorner().getY() + "))";

        query.setString(1, lineGeom);
        query.setString(2, polyGeom);
        query.setString(3, polyGeom);
        query.setString(4, polyGeom);
        query.setString(5, polyGeom);
        query.setString(6, polyGeom);

        rs = query.executeQuery();

```

The `BallonTemplateHandlerImpl` class builds up a queries for the KML balloon content. Most of them are aggregated queries. If multiple rows are fetched by the `ResultSet` and no aggregation was used one row has to be picked. Therefore the window function `ROW_NUMBER()` was used. As *PostgreSQL* does not allow the usage of window function inside of a `WHERE` clause the queries have to be re-written in a more nested way (except for the first example, that did not need a range condition for `rnum` like in *Oracle*).

de.tub.citydb.modules.kml.database.**BallonTemplateHandlerImpl**

```

1152  sqlStatement = sqlStatement + ") AS subquery"; // PostgreSQL-Query needs
rep                                     an alias here

1206  //      sqlStatement = "SELECT * FROM " +
//          " (SELECT a.*, ROWNUM rnum FROM (" + sqlStatement +
//          " ORDER by " + tableShortId + "." + columns.get(0) + " ASC) a"
//          + " WHERE ROWNUM <= " + rownum + ") "
//          + "WHERE rnum >= " + rownum;
sqlStatement = "SELECT * FROM " +
    "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC) AS rnum" +
    " FROM (" + sqlStatement +
    " ORDER BY " + tableShortId + "." + columns.get(0) + " ASC) sqlstat)
    AS subq WHERE rnum = " + rownum;

//      else if (FIRST.equalsIgnoreCase(aggregateFunction)) {
//          sqlStatement = "SELECT * FROM (" + sqlStatement +
//          " ORDER by " + tableShortId + "." + columns.get(0) + " ASC)" +
//          " WHERE ROWNUM = 1";
//      }
//      else if (LAST.equalsIgnoreCase(aggregateFunction)) {
//          sqlStatement = "SELECT * FROM (" + sqlStatement +
//          " ORDER by " + tableShortId + "." + columns.get(0) + " DESC)"
//          + " WHERE ROWNUM = 1";

```

```

//      }
else if (FIRST.equalsIgnoreCase(aggregateFunction)) {
    sqlStatement = "SELECT * FROM " +
        "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC)
        AS rnum FROM (" + sqlStatement +
        " ORDER BY " + tableShortId + "." + columns.get(0) + " ASC) sqlstat)
        AS subq WHERE rnum = 1";
}
else if (LAST.equalsIgnoreCase(aggregateFunction)) {
    sqlStatement = "SELECT * FROM " +
        "(SELECT sqlstat.*, ROW_NUMBER() OVER(ORDER BY sqlstat.* ASC)
        AS rnum FROM (" + sqlStatement +
        " ORDER BY " + tableShortId + "." + columns.get(0) + " DESC)
        sqlstat) AS subq WHERE rnum = 1";
}

```

## 9.2 Geometries for KML placemarks

Most of the changes were similar to examples in chapter 5 and more or less self-explaining. The `JGeometry.getOrdinatesArray()` method is substituted with a simple iteration to fill the array. Some extra variables and *PostGIS* JDBC classes (and their methods) are used to port *Oracle*'s ELEM-INFO methods correctly.

de.tub.citydb.modules.kml.database.**CityObjectGroup**

```

189  //      STRUCT buildingGeometryObj = (STRUCT)rs.getObject(1);
      PGgeometry pgBuildingGeometry = (PGgeometry)rs.getObject(1);

200  //      JGeometry groundSurface =
rep+ //      convertToWGS84(JGeometry.load(buildingGeometryObj));
      //      int dim = groundSurface.getDimensions();
      //      for (int i = 0; i < groundSurface.getElemInfo().length; i = i+3) {
      //          LinearRingType linearRing = kmlFactory.createLinearRingType();
      //          BoundaryType boundary = kmlFactory.createBoundaryType();
      //          boundary.setLinearRing(linearRing);
      //          switch (groundSurface.getElemInfo()[i+1]) {
      //              case EXTERIOR_POLYGON_RING: // = 1003
      //                  polygon.setOuterBoundaryIs(boundary);
      //                  break;
      //              case INTERIOR_POLYGON_RING: // = 2003
      //                  polygon.getInnerBoundaryIs().add(boundary);
      //                  break;
      //              case POINT: // = 1
      //              case LINE_STRING: // = 2
      //                  continue;
      //              default:
      //                  Logger.getInstance().warn("Unknown
      //                      geometry for " + work.getGmlId());
      //                  continue;
      //          }
      //      }
      //      double[] ordinatesArray = groundSurface.getOrdinatesArray();
      //      int startNextGeometry = ((i+3) < groundSurface.getElemInfo().length) ?
      //          groundSurface.getElemInfo()[i+3] - 1: // still more geometries
      //          ordinatesArray.length; // default

```

```

//
//      // order points counter-clockwise
//      for (int j = startNextGeometry - dim;
//          j >= groundSurface.getElemInfo()[i] - 1; j = j dim) {
//          linearRing.getCoordinates().add(String.valueOf(
//              ordinatesArray[j] + "," + ordinatesArray[j+1] + ",0"));
//      }
Geometry groundSurface = convertToWGS84(pgBuildingGeometry.getGeometry());

switch (groundSurface.getType()) {
case Geometry.POLYGON:
    Polygon polyGeom = (Polygon)groundSurface;

    for (int ring = 0; ring < polyGeom.numRings(); ring++){
        LinearRingType linearRing = kmlFactory.createLinearRingType();
        BoundaryType boundary = kmlFactory.createBoundaryType();
        boundary.setLinearRing(linearRing);

        double [] ordinatesArray =
            new double[polyGeom.getRing(ring).numPoints()*2];

        for (int j=polyGeom.getRing(ring).numPoints()-1,k=0;
            j>=0;j--,k+=2){
            ordinatesArray[k] = polyGeom.getRing(ring).getPoint(j).x;
            ordinatesArray[k+1] = polyGeom.getRing(ring).getPoint(j).y;
        }

        // the first ring usually is the outer ring in a PostGIS-Polygon
        // e.g. POLYGON((outerBoundary), (innerBoundary), (innerBoundary))
        if (ring == 0){
            polygon.setOuterBoundaryIs(boundary);
            for (int j = 0; j < ordinatesArray.length; j+=2) {
                linearRing.getCoordinates().add
                    (String.valueOf(ordinatesArray[j] + "," +
                        ordinatesArray[j+1] + ",0"));
            }
        }
        else {
            polygon.getInnerBoundaryIs().add(boundary);
            for (int j = ordinatesArray.length - 2; j >= 0; j-=2) {
                linearRing.getCoordinates().add(
                    String.valueOf(ordinatesArray[j] + "," +
                        ordinatesArray[j+1] + ",0"));
            }
        }
    }
    break;
case Geometry.MULTIPOLYGON:
    MultiPolygon multiPolyGeom = (MultiPolygon) groundSurface;
    multiPolygon = new PolygonType[multiPolyGeom.numPolygons()];

    for (int p = 0; p < multiPolyGeom.numPolygons(); p++){
        Polygon subPolyGeom = multiPolyGeom.getPolygon(p);

        multiPolygon[p] = kmlFactory.createPolygonType();
        multiPolygon[p].setTessellate(true);
        multiPolygon[p].setExtrude(true);
    }
}

```



```

        multiPolygon[p].setAltitudeModeGroup
            (kmlFactory.createAltitudeMode(
                AltitudeModeEnumType.
                    RELATIVE_TO_GROUND));

    for (int ring = 0; ring < subPolyGeom.numRings(); ring++){
        LinearRingType linearRing = kmlFactory.createLinearRingType();
        BoundaryType boundary = kmlFactory.createBoundaryType();
        boundary.setLinearRing(linearRing);

        double [] ordinatesArray = new
        double[subPolyGeom.getRing(ring).numPoints() * 2];
        for (int j=subPolyGeom.getRing(ring).numPoints()-1, k=0;
            j >= 0; j--, k+=2){
            ordinatesArray[k] =
                subPolyGeom.getRing(ring).getPoint(j).x;
            ordinatesArray[k+1] =
                subPolyGeom.getRing(ring).getPoint(j).y;
        }

        // the first ring usually is the outer ring in a PostGIS-
        // Polygon e.g. POLYGON((outerBoundary),(innerBoundary),
        // (innerBoundary))
        if (ring == 0){
            multiPolygon[p].setOuterBoundaryIs(boundary);
            for (int j = 0; j < ordinatesArray.length; j+=2) {
                linearRing.getCoordinates().add(
                    String.valueOf(ordinatesArray[j] + "," +
                        ordinatesArray[j+1] + ",0"));
            }
        }
        else {
            multiPolygon[p].getInnerBoundaryIs().add(boundary);
            for (int j = ordinatesArray.length - 2; j >= 0; j-=2) {
                linearRing.getCoordinates().add(
                    String.valueOf(ordinatesArray[j] + "," +
                        ordinatesArray[j+1] + ",0"));
            }
        }
    }
}

case Geometry.POINT:
case Geometry.LINESTRING:
case Geometry.MULTIPOINT:
case Geometry.MULTILINESTRING:
case Geometry.GEOMETRYCOLLECTION:
    continue;
default:
    Logger.getInstance().warn("Unknown geometry for " +
        work.getGmlId());
    continue;
}
}
if (polygon != null){
    multiGeometry.getAbstractGeometryGroup().add(
        kmlFactory.createPolygon(polygon));
}

```

```

    if (multiPolygon != null){
        for (int p = 0; p < multiPolygon.length; p++){
            multiGeometry.getAbstractGeometryGroup().add(
                kmlFactory.createPolygon(multiPolygon[p]));
        }
    }
}

```

## de.tub.citydb.modules.kml.database.KmlGenericObject

```

2031 //      STRUCT buildingGeometryObj = (STRUCT)rs.getObject(1);
rep //      JGeometry surface =
//          convertToWGS84(JGeometry.load(buildingGeometryObj));
//      double[] ordinatesArray = surface.getOrdinatesArray();
PGGeometry pgBuildingGeometry = (PGGeometry)rs.getObject(1);
Polygon surface =
    (Polygon)convertToWGS84(pgBuildingGeometry.getGeometry());

double[] ordinatesArray = new double[surface.numPoints()*3];

for (int i = 0, j = 0; i < surface.numPoints(); i++, j+=3){
    ordinatesArray[j] = surface.getPoint(i).x;
    ordinatesArray[j+1] = surface.getPoint(i).y;
    ordinatesArray[j+2] = surface.getPoint(i).z;
}

2064 //      for (int i = 0; i < surface.getElemInfo().length; i = i+3) {
rep //          LinearRingType linearRing = kmlFactory.createLinearRingType();
//          BoundaryType boundary = kmlFactory.createBoundaryType();
//          boundary.setLinearRing(linearRing);
//          if (surface.getElemInfo()[i+1] == EXTERIOR_POLYGON_RING) {
//              polygon.setOuterBoundaryIs(boundary);
//          }
//          else { // INTERIOR_POLYGON_RING
//              polygon.getInnerBoundaryIs().add(boundary);
//          }
//          int startNextRing = ((i+3) < surface.getElemInfo().length) ?
//              surface.getElemInfo()[i+3] - 1: // still holes to come
//              ordinatesArray.length; // default
//          // order points clockwise
//          for (int j = surface.getElemInfo()[i] - 1; j < startNextRing;
//              j = j+3) {
//              linearRing.getCoordinates().add(
//                  String.valueOf(
//                      reducePrecisionForXorY(ordinatesArray[j]) + "," +
//                      reducePrecisionForXorY(ordinatesArray[j+1]) + "," +
//                      reducePrecisionForZ(ordinatesArray[j+2] +
//                          zOffset));
//              probablyRoof = ...
//          }
int cellCount = 0; // equivalent to first value of Oracle's SDO_ELEM_INFO

for (int i = 0; i < surface.numRings(); i++){
    LinearRingType linearRing = kmlFactory.createLinearRingType();
    BoundaryType boundary = kmlFactory.createBoundaryType();
    boundary.setLinearRing(linearRing);
}

```

```

        if (i == 0) { // first ring is the outer ring
            polygon.setOuterBoundaryIs(boundary);
        } else {
            polygon.getInnerBoundaryIs().add(boundary);
        }
        int startNextRing = ((i+1) < surface.numRings()) ?
            (surface.getRing(i).numPoints()*3): // still holes to come
            ordinatesArray.length; // default
        // order points clockwise
        for (int j = cellCount; j < startNextRing; j+=3 {
            linearRing.getCoordinates().add(
                String.valueOf(
                    reducePrecisionForXorY(ordinatesArray[j]) + "," +
                    reducePrecisionForXorY(ordinatesArray[j+1]) + "," +
                    reducePrecisionForZ(ordinatesArray[j+2] + zOffset))
                probablyRoof = ...
            )
        }
        cellCount += (surface.getRing(i).numPoints()*3);
    }

2540 // int contourCount = unconvertedSurface.getElemInfo().length/3;
    // // remove normal-irrelevant points
    // int startContour1 = unconvertedSurface.getElemInfo()[0] - 1;
    // int endContour1 = (contourCount == 1) ?
    //     ordinatesArray.length: // last
    //     unconvertedSurface.getElemInfo()[3] - 1; // holes are irrelevant
    //                                     for normal calculation
    // // last point of polygons in gml is identical to first and useless
    // // for GeometryInfo
    // endContour1 = endContour1 - 3;
    int contourCount = unconvertedSurface.numRings();
    int startContour1 = 0;
    int endContour1 = (contourCount == 1) ?
        ordinatesArray.length: // last
        (unconvertedSurface.getRing(startContour1).numPoints()*3);
    endContour1 = endContour1 - 3;

2586 // for (int i = 0; i < ordinatesArray.length; i = i + 3) {
    //     // coordinates = coordinates + h1Distance * (dot product of normal
    //     // vector and unity vector)
    //     ordinatesArray[i] = ordinatesArray[i] + h1Distance * nx;
    //     ordinatesArray[i+1] = ordinatesArray[i+1] + h1Distance * ny;
    //     ordinatesArray[i+2] = ordinatesArray[i+2]+zOffset+h1Distance*nz;
    // }

    for (int i = 0, j = 0; i < unconvertedSurface.numPoints(); i++, j+=3){
        unconvertedSurface.getPoint(i).x = ordinatesArray[j] + h1Distance*nx;
        unconvertedSurface.getPoint(i).y = ordinatesArray[j+1] + h1Distance*ny;
        unconvertedSurface.getPoint(i).z = ordinatesArray[j+2] + zOffset +
            h1Distance * nz;
    }

```

## 9.3 Textures for COLLADA-Export

The database can store texture formats that are unknown to ORDImage. Therefore two methodologies were implemented in the *KML-Exporter*. One to deal with ORDImages and another to process all the unknown formats as BLOBs. Fortunately the last one could be used for the *PostGIS* port. All the TexOrdImage methods had to be uncommented from the following classes and the texture-export for COLLADA exports was slightly changed.

de.tub.citydb.modules.kml.database.**KmlGenericObject**

```

2238 //      OrdImage texImage = null;
        InputStream texImage = null;

2262 addTexImageUri(surfaceId, texImageUri);
        // if (getTexOrdImage(texImageUri) == null) { // not already marked as
                                                wrapping texture

2283 //bufferedImage = ImageIO.read(texImage.getDataInStream());
        bufferedImage = ImageIO.read(texImage);

2290 // else {
        //      addTexOrdImage(texImageUri, texImage);
        //      }
        // }

2256 // texture wrapping -- it conflicts with texture atlas
        removeTexImage(texImageUri);
        BufferedImage bufferedImage = null;
        try {
            bufferedImage = ImageIO.read(texImage);
        } catch (IOException e) {}
        addTexImage(texImageUri, bufferedImage);
        //      addTexOrdImage(texImageUri, texImage);

```

de.tub.citydb.modules.kml.concurrent.**KmlExportWorker**

de.tub.citydb.modules.kml.controller.**KmlExporter**

de.tub.citydb.modules.kml.database.**ColladaBundle**

de.tub.citydb.modules.kml.database.**KmlExporterManager**

```

rep+ //      uncommented TexOrdImage-methods

```