virtual city systems

digital views. real perspectives.

3D City Database

Implementing an Importer/Exporter ADE Extension

Zhihang Yao, zyao@vc.systems

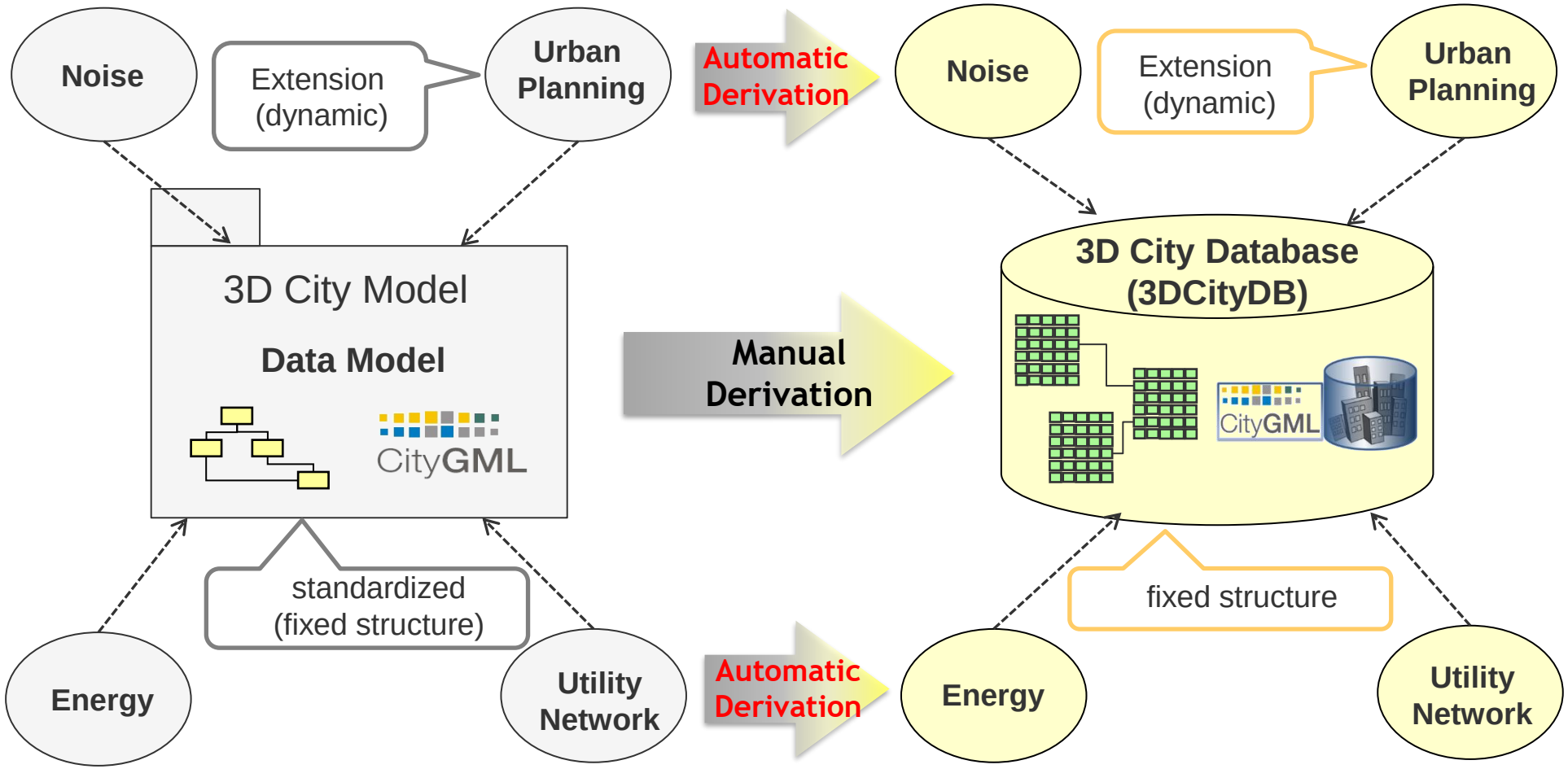
Claus Nagel, cnagel@vc.systems

General introduction

Application Domain Extension (ADE)

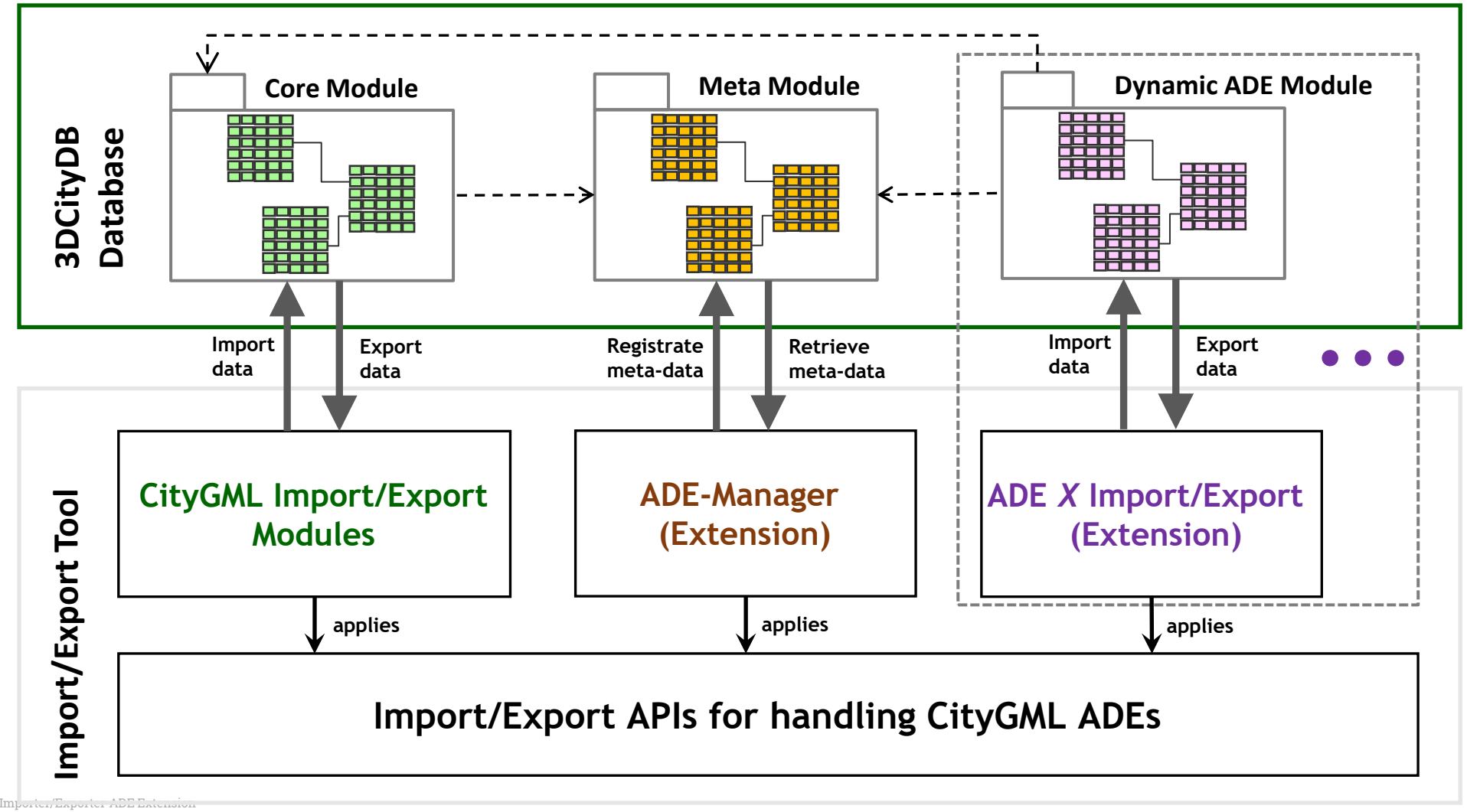
- **Extension** of the CityGML model defined by **information communities** for specific application domains
- **Formal specification** in separate XML schema documents referencing the CityGML schemas
- Generally, two types of domain specific extensions can be distinguished:
  - Extension of existing CityGML feature types
    - Additional spatial and non-spatial properties
    - Additional relations / associations to other feature types
  - Definition of new feature and complex data types

Automatic Derivation of DB-Schemas for ADEs



Object-oriented model **Spatio-relational model**

Extending the 3DCityDB to support ADEs



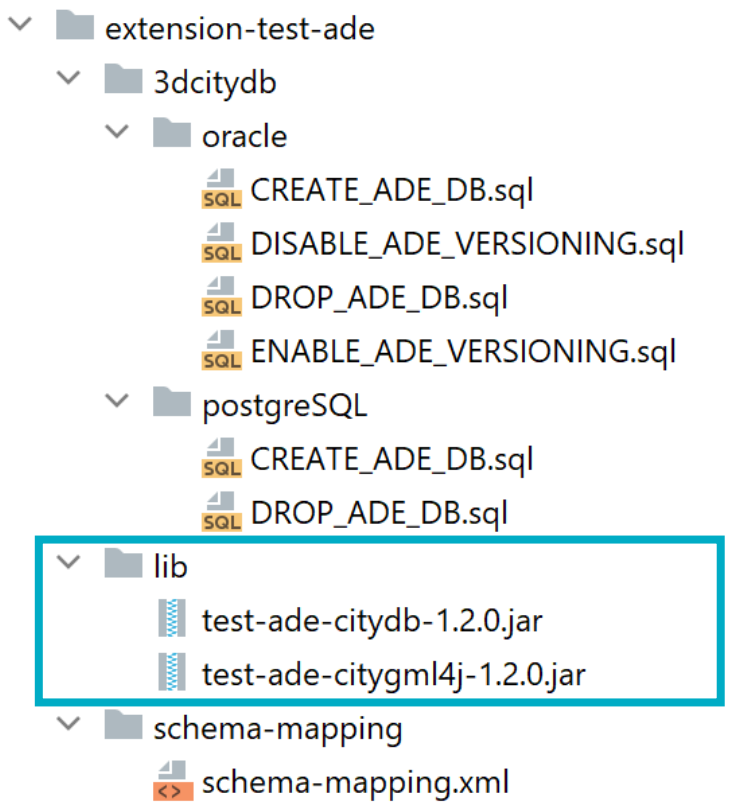
3DCityDB ADE Extension Package

1. Database schema extension

- SQL scripts to create and drop the ADE-specific tables
- Schema mapping file (XML schema to relational schema and metadata)
- Can be automatically created with ADE Manager Plugin

2. Importer/Exporter extension

- Implement ADE interface to handle ADE datasets
- CityGML import and export (requires a citygml4j ADE module)
- KML/COLLADA/glTF export
- Spreadsheet export for attributes



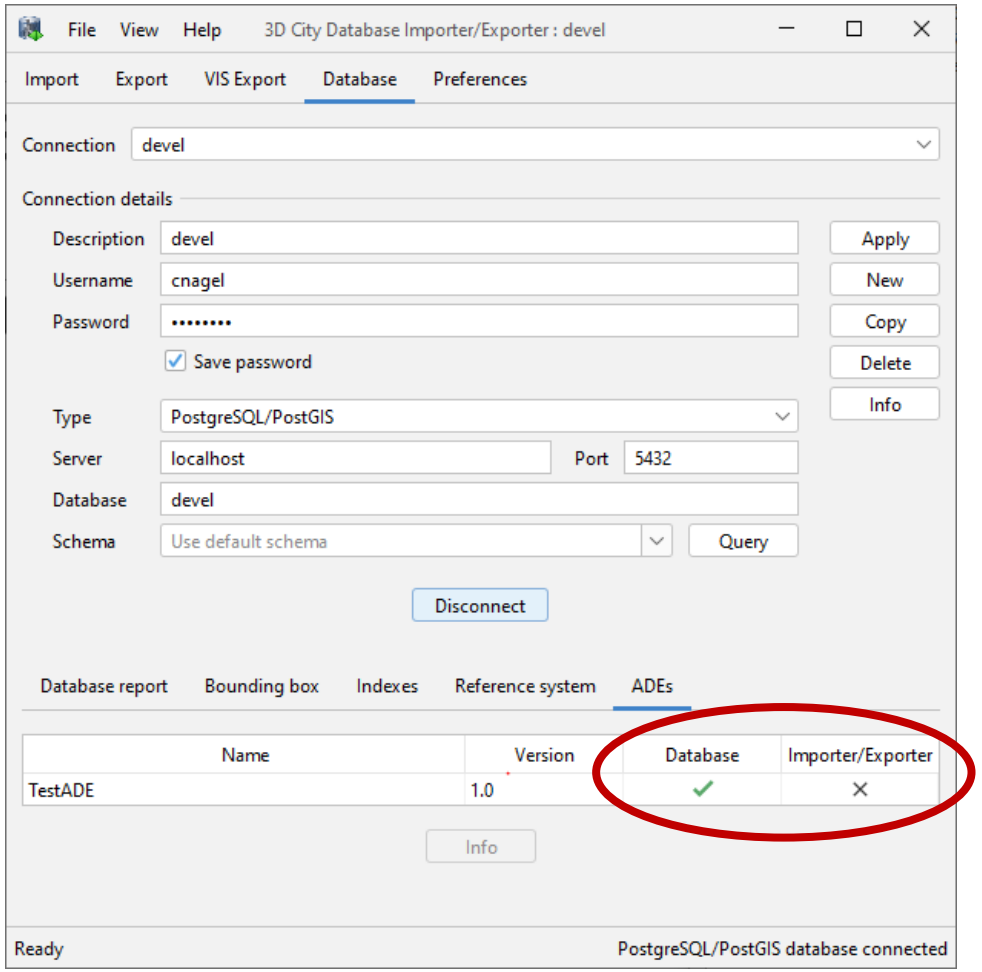
© VC Systems 2021 / Implementing an Importer/Exporter ADE Extension

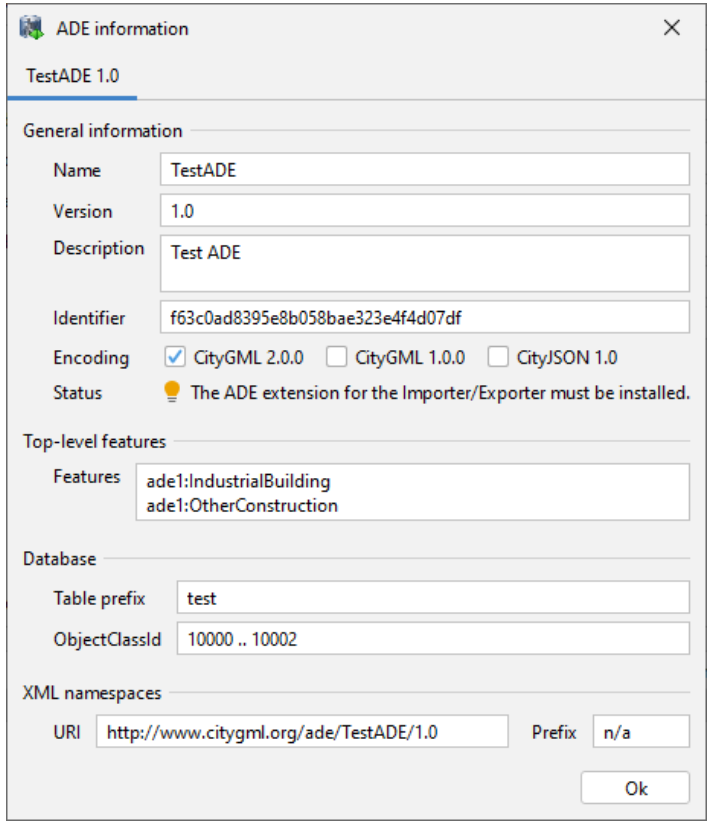
Create and register your Database ADE Schema Extension

ADE Manager Plugin

- Use the ADE Manager Plugin to create a Database Schema Extension
  - Input: ADE XML Schema
  - Output: SQL scripts and an schema-mapping.xml file
- Use the ADE Manager Plugin to register the Database Schema Extension with your 3DCityDB instance
  - ADE schema (tables, sequences, FK constraints, indexes, …) is created, object class IDs are registered, …
- Documentation
  - https://3dcitydb-docs.readthedocs.io/en/version-2021.1/plugins/ade-manager/index.html

After this step, your ADE schema should be available…





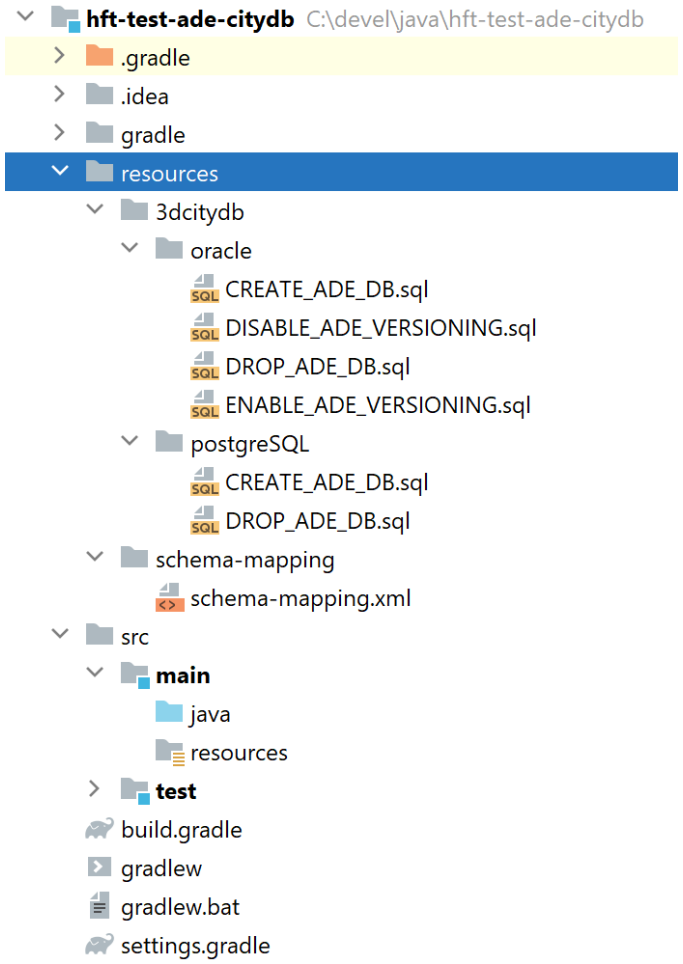
First steps to implementing an Importer/Exporter ADE Extension

Step 1: Set up development environment

- Add the following Maven repositories
  - https://citydb.jfrog.io/artifactory/maven (Importer/Exporter libraries)
  - https://repo.osgeo.org/repository/release (GeoTools libraries)
- Add a dependency to the Maven artifact org.citydb:impexp-client-gui:<version>
  - Replace <version> with your target version (latest version: 5.0.0)
  - impexp-core would be sufficient as dependency but impexp-client-gui is **recommended** for easy testing
- Copy the ADE database schema extension to your project
- Make sure to include your citygml4j ADE module
  - Either as external library or as part of the source code

Example Gradle configuration and IntelliJ project

```
plugins {
    id 'java-library'
}

group 'org.example'
version '1.0-SNAPSHOT'

java {
    sourceCompatibility = JavaVersion.VERSION_1_8
}

repositories {
    maven {
        url 'https://citydb.jfrog.io/artifactory/maven'
    }
    maven {
        url 'https://repo.osgeo.org/repository/release'
    }
    mavenCentral()
}

dependencies {
    api 'org.citydb:impexp-client:4.3.0'
}
```



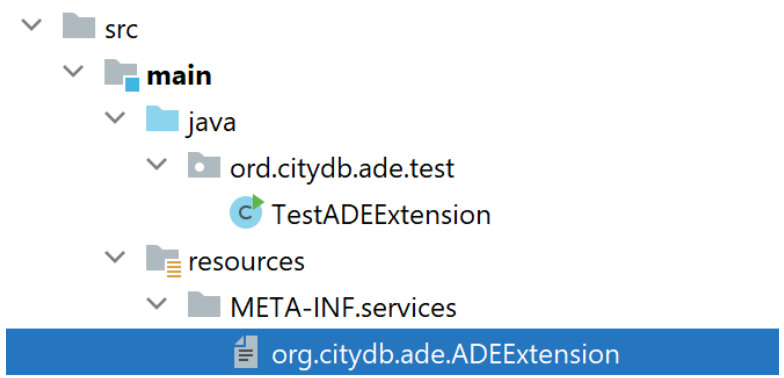
Step 2: Implement ADEExtension class skeleton

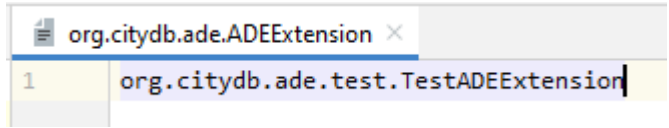
```
public class TestADEExtension extends ADEExtension {


    @Override
    public void init(SchemaMapping schemaMapping) throws ADEExtensionException {

    }

    @Override
    public List<ADEContext> getADEContexts() {
        return null;
    }

    @Override
    public ADEObjectMapper getADEObjectMapper() {
        return null;
    }

    @Override
    public ADEImportManager createADEImportManager() {
        return null;
    }

    @Override
    public ADEExportManager createADEExportManager() {
        return null;
    }
}
```

- **CityGML ADE import/export** requires implementing the abstract class ADEExtension

- init(SchemaMapping)
  - Initialize your extension
- getADEContexts()
  - Return the citygml4j ADE context(s)
- getADEObjectMapper()
  - Implement and return an ADEObjectMapper
- createADEImportManager()
  - Implement and return an ADEImportManager
- createADEExportManager()
  - Implement and return an ADEExportManager

Step 3: Add your class to META-INF/services





- Create a file called org.citydb.ade.ADEExtension in META-INF/services
- Add the fully qualified name of your class from step 2 on a single line in this file
- Make sure that this file is included at the correct place when building the JAR

- This way, your class is registered as Service Provider for the ADEExtension class
- The Importer/Exporter will automatically load your ADE extension using the services information from the JAR

Step 4: Add a main method for easy testing

```
public class TestADEExtension extends ADEExtension {

    public static void main(String[] args) {
        TestADEExtension extension = new TestADEExtension();
        extension.setBasePath(Paths.get("resources").toAbsolutePath());

        new ImpExpLauncher()
                .withArgs(args)
                .withADEExtension(extension)
                .start();
    }

    ...
}
```

- Use ImpExpLauncher to start the GUI of the Importer/Exporter
- Register your ADE extension using the withADEExtension method
  - Only for testing; in the end, the Java Service Loader is used for this (see step 3)
- Make sure the database schema extension can be loaded **by setting the base path** to the corresponding folder inside your project (see step 1)

© VC Systems 2021 / Implementing an Importer/Exporter ADE Extension

Step 5: Complete minimal ADEExtension implementation

- Implement getADEContexts() method
- Implement an ADEObjectMapper
  - Maps a database objectclass_id to corresponding citygml4j object and vice versa
- Implement ADEImportManager skeleton
  - Handles the import of ADE data
- Implement ADEExportManager skeleton
  - Handles the export of ADE data

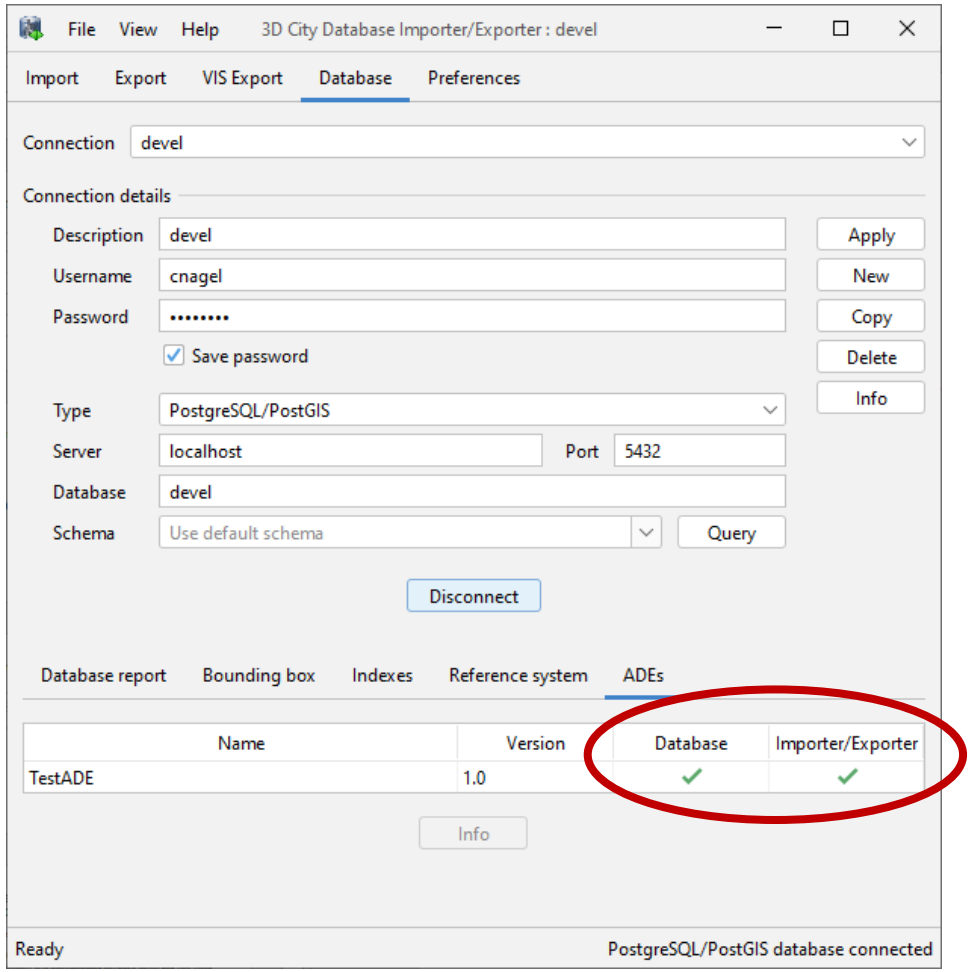
ADEObjectMapper skeleton

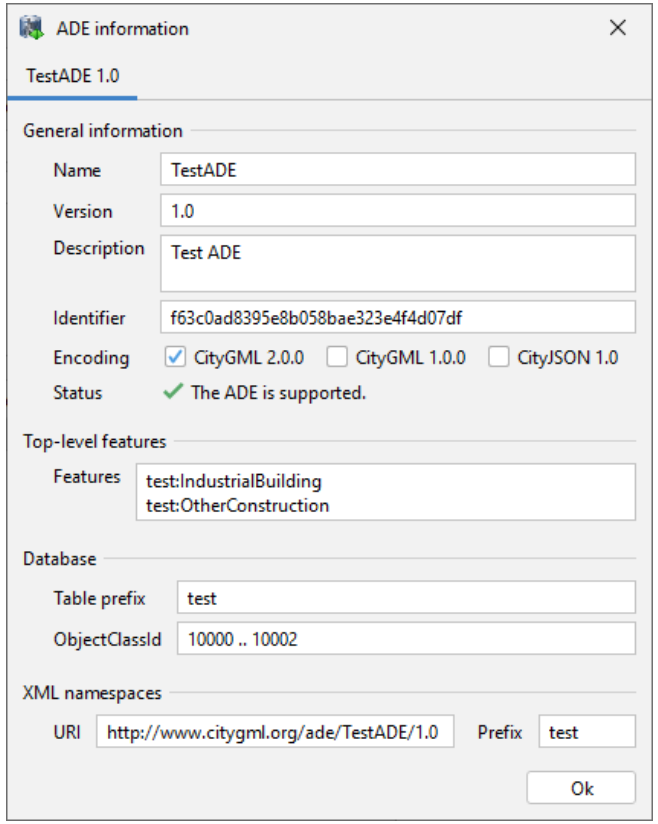
```
public class ObjectMapper implements ADEObjectMapper {

    @Override
    public AbstractGML createObject(int objectClassId, CityGMLVersion version) {
        return null;
    }

    @Override
    public int getObjectClassId(Class<? extends AbstractGML> adeObjectClass) {
        return 0;
    }
}
```

```
<featureType id="test_IndustrialBuildingType" table="test_industrialbuilding"
 objectClassId="10001" topLevel="true" path="IndustrialBuilding" schema="test">
    <extension base="AbstractBuildingType">
        <join table="building" fromColumn="id" toColumn="id" toRole="parent"/>
    </extension>
    <attribute column="remark" type="string" maxOccurs="1" path="remark" schema="test"/>
</featureType>
```

- Maps between object class IDs and citygml4j objects
- Object class IDs are assigned to object types in the schema mapping file
- Mapping can be hard-coded or realized by scanning the schema mapping file (recommended)

After step 5, your Importer/Exporter ADE extension should be available…





Implement the ADEImportManager interface to import ADE data

ADEImportManager skeleton (1/4)

```
public class ImportManager implements ADEImportManager {

    @Override
    public void init(Connection connection, CityGMLImportHelper helper) throws CityGMLImportException, SQLException {

    }

    @Override
    public void importObject(ADEModelObject object, long objectId, AbstractObjectType<?> objectType, ForeignKeys foreignKeys)
                            throws CityGMLImportException, SQLException {

    }

    @Override
    public void importGenericApplicationProperties(ADEPropertyCollection properties, AbstractFeature parent, long parentId, FeatureType parentType)
                                                   throws CityGMLImportException, SQLException {

    }

    @Override
    public void executeBatch(String tableName) throws CityGMLImportException, SQLException {

    }

    @Override
    public void close() throws CityGMLImportException, SQLException {

    }
}
```

ADEImportManager skeleton (2/4)

- importObject method
  - Invoked for any object defined in your ADE and found in the dataset
  - You must **only import ADE-specific properties** defined for the object
  - Non-ADE properties (i.e., properties inherited from CityGML object types) **are managed by the Importer/Exporter**
  - There are no restrictions on the way you use JDBC

- Input
  - ADEModelObject object: the citygml4j object as defined in your citygml4j ADE module
  - long objectId: the assigned database ID
  - AbstractObjectType<?> objectType: the object type from the schema mapping file
  - ForeignKeys foreignKeys: additional foreign keys (empty unless populated by you)

ADEImportManager skeleton (3/4)

- importGenericApplicationProperties method
  - Invoked for „ADE hook properties“ defined in your ADE and found in the dataset
  - All ADE properties assigned to the same feature are passed in one call
  - You must **only import the ADE hook properties** (not the parent feature)
  - There are no restrictions on the way you use JDBC

- Input
  - ADEPropertyCollection properties: The set of ADE properties
  - AbstractFeature parent: The feature to which the ADE properties are assigned
  - long parentId: The database ID of the feature
  - FeatureType parentType: the feature type from the schema mapping file

ADEImportManager skeleton (4/4)

- executeBatch(String tableName) method
  - Start the execution of all batched import statements for the given ADE table
  - It is **strongly recommended** but not mandatory that you use batch processing
  - Invoked for all your ADE tables at least once during an import session

- close method
  - Close all database (and possibly further) resources that you have opened in your own importer classes
  - Do not close resources passed to your classes; they are managed by the Importer/Exporter
  - Invoked at the end of the import session

Import utility classes making your life easier

CityGMLImportHelper utility (1/5)

```
public interface CityGMLImportHelper {
   public long importObject(AbstractGML object) throws CityGMLImportException, SQLException;
   public long importObject(AbstractGML object, ForeignKeys foreignKeys) throws CityGMLImportException, SQLException;
   public long importGlobalAppearance(Appearance appearance) throws CityGMLImportException, SQLException;
   public long importSurfaceGeometry(AbstractGeometry surfaceGeometry, long cityObjectId) throws CityGMLImportException, SQLException;
   public long importImplicitGeometry(ImplicitGeometry implicitGeometry) throws CityGMLImportException, SQLException;
   public GeometryConverter getGeometryConverter();
   public String convertImplicitGeometryTransformationMatrix(TransformationMatrix4x4 matrix);
   public boolean isSurfaceGeometry(AbstractGeometry abstractGeometry);
   public boolean isPointOrLineGeometry(AbstractGeometry abstractGeometry);

   public AbstractDatabaseAdapter getDatabaseAdapter();
   public void executeBatch(String tableName) throws CityGMLImportException, SQLException;
   public void executeBatch(AbstractObjectType<?> type) throws CityGMLImportException, SQLException;
   public String getTableNameWithSchema(String tableName);
   public long getNextSequenceValue(String sequence) throws SQLException;
   public AttributeValueJoiner getAttributeValueJoiner();

   public boolean isFailOnError();
   public ImportConfig getImportConfig();

   public void logOrThrowUnsupportedXLinkMessage(AbstractGML from, Class<? extends AbstractGML> to, String xlink) throws CityGMLImportException;
   public void logOrThrowUnsupportedGeometryMessage(AbstractGML from, AbstractGeometry geometry) throws CityGMLImportException;
   public void logOrThrowErrorMessage(String message) throws CityGMLImportException;
   public String getObjectSignature(AbstractGML object);

   public int getObjectClassId(AbstractGML object);
   public FeatureType getFeatureType(AbstractFeature feature);
   public ObjectType getObjectType(AbstractGML object);
   public AbstractObjectType<?> getAbstractObjectType(AbstractGML object);

   public void propagateObjectXlink(String table, long objectId, String xlink, String propertyColumn);
   public void propagateObjectXlink(String intermediateTable, long objectId, String fromColumn, String xlink, String toColumn);
   public void propagateReverseObjectXlink(String toTable, String gmlId, long objectId, String propertyColumn);
   public void propagateSurfaceGeometryXlink(String xlink, String table, long objectId, String propertyColumn);
}
```

CityGMLImportHelper utility (2/5)

```
public long importObject(AbstractGML object);
public long importObject(AbstractGML object, ForeignKeys foreignKeys);
public long importGlobalAppearance(Appearance appearance);
public long importSurfaceGeometry(AbstractGeometry surfaceGeometry,
            long cityObjectId);
public long importImplicitGeometry(ImplicitGeometry implicitGeometry);
public GeometryConverter getGeometryConverter();
public String convertImplicitGeometryTransformationMatrix(
              TransformationMatrix4x4 matrix);
public boolean isSurfaceGeometry(AbstractGeometry abstractGeometry);
public boolean isPointOrLineGeometry(
               AbstractGeometry abstractGeometry);
```

- importObject methods
  - Used to push a GML object to the Importer/Exporter for import
  - If the object is from your ADE, the Importer/Exporter **will call back** your ADEImportManager

- importSurfaceGeometry method
  - Used to import surface-based GML geometries into SURFACE_GEOMETRY

- GeometryConverter
  - Used to convert a GML geometry into a representation that can be imported into a geometry column

- Helper methods to import implicit geometries into the 3DCityDB-specific representation

CityGMLImportHelper utility (3/5)

```
public AbstractDatabaseAdapter getDatabaseAdapter();
public void executeBatch(String tableName);
public void executeBatch(AbstractObjectType<?> type);
public String getTableNameWithSchema(String tableName);
public long getNextSequenceValue(String sequence);
public AttributeValueJoiner getAttributeValueJoiner();
```

- executeBatch methods
  - Used to execute batched statements for a specific object type or table
  - **Always use these methods** because the Importer/Exporter will handle execution order for you (e.g., to ensure FK constraints)

- getNextSequenceValue method
  - If your relational ADE schema contains additional sequences, use this method to get the next value (abstracts from the underlying database system)

- getTableNameWithSchema method
  - Adds the schema name of the database connection as prefix to your table name (“schema.tableName”)

- AttributeValueJoiner
  - Used to join multiple values into a String using the 3DCityDB-specific delimiter “--/\--”

CityGMLImportHelper utility (4/5)

```
public boolean isFailOnError();
public ImportConfig getImportConfig();

public void logOrThrowUnsupportedXLinkMessage(AbstractGML from,
            Class<? extends AbstractGML> to, String xlink);
public void logOrThrowUnsupportedGeometryMessage(AbstractGML from,
            AbstractGeometry geometry);
public void logOrThrowErrorMessage(String message);
public String getObjectSignature(AbstractGML object);

public int getObjectClassId(AbstractGML object);
public FeatureType getFeatureType(AbstractFeature feature);
public ObjectType getObjectType(AbstractGML object);
public AbstractObjectType<?> getAbstractObjectType(AbstractGML object);
```

- ImportConfig
  - Get access to all import preference settings made by the user for the current import session

- Logging helpers
  - Some helper methods to create standardized log messages
  - getObjectSignature method turns an object into a String representation that can be used in log messages

- SchemaMapping helpers
  - Some helper methods to get the object class ID for a given object or its type definition from the schema mapping

CityGMLImportHelper utility (5/5)

```
public void propagateObjectXlink(String table, long objectId, String xlink, String propertyColumn);
public void propagateObjectXlink(String intermediateTable, long objectId, String fromColumn, String xlink, String toColumn);
public void propagateReverseObjectXlink(String toTable, String xlink, long objectId, String propertyColumn);
public void propagateSurfaceGeometryXlink(String xlink, String table, long objectId, String propertyColumn);
```

- propagateObjectXlink method
  - Insert the database ID of the object referenced by the XLink into the specified table and column at the specified record (objectId)
  - The table can be a feature table or an intermediate (n:m) table
- propagateReverseObjectXlink method
  - Insert the database ID (objectId) of the current object into the specified table and column of the object referenced by the XLink
- propagateSurfaceGeometryXlink method
  - Insert the database ID of the geometry referenced by the XLink into the specified table and column at the specified record (objectId)

Database Adapter (1/2)

- Used to get **metadata** about the currently connected database and to **execute operations** on this database
- Abstracts from the underlying database system (PostgreSQL / Oracle)

- Metadata includes
  - Connection details (database type and name, host, port, username, etc.)
  - Details about 3DCityDB instance like version, reference system, etc.
  - Details about underlying database system (product name and version, etc.)
  - SQL/JDBC specific details (maximum batch size, maximum size for IN operator, etc.)
  - …

- Operations include
  - Get sequence values, transform geometries, manage indexes, create database report, etc.

Database Adapter (2/2)

Most important functionality when importing ADE data:

- getMaxBatchSize method
  - Get the database-specific maximum number of statements allowed in one batch
  - If the number is reached, the batch must be executed

- GeometryConverter
  - Used to create a database-specific geometry object or NULL value to be stored in a geometry column

Further interfaces and utilities (1/2)

- ForeignKeys store
  - Assume you have two ADE city objects A and B. B is a nested object of A and the table of B has a column of name „ref_to_a“ which is a foreign key to its parent object A.
  - When importing object A you want to use the importObject method of CityGMLImportHelper to make sure that CityGML-specific stuff (like appearances) of object B is handled by the Importer/Exporter.
    - So, the Importer/Exporter will call back your code for importing object B
    - But in this callback, you will have lost the current information about object A, such as its database ID which is needed for the „ref_to_a“ column of object B
  - To solve this issue, you can store the database ID of object A in the ForeignKeys store and pass it to the importObject method together with object B
    - Simple name/value store (for example, use „ref_to_a“ as name and the ID of object A as value)
    - When the Importer/Exporter calls your code for importing object B, it will also pass your ForeignKeys store with all the information you need to populate the „ref_to_a“ column

Further interfaces and utilities (2/2)

- ADEImporter interface
  - Simple interface that you can use for the import classes working on different citygml4j objects and tables
  - Optional but helpful

Implement the ADEExportManager interface to export ADE data

ADEExportManager skeleton (1/4)

```
public class ExportManager implements ADEExportManager {

    @Override
    public void init(Connection connection, CityGMLExportHelper helper) throws CityGMLExportException, SQLException {

    }

    @Override
    public void exportObject(ADEModelObject object, long objectId, AbstractObjectType<?> objectType, ProjectionFilter projectionFilter)
                            throws CityGMLExportException, SQLException {

    }

    @Override
    public void exportGenericApplicationProperties(String adeHookTable, AbstractFeature parent, long parentId, FeatureType parentType,
                                                   ProjectionFilter projectionFilter)
                                                   throws CityGMLExportException, SQLException {

    }

    @Override
    public void close() throws CityGMLExportException, SQLException {

    }
}
```

ADEExportManager skeleton (2/4)

- exportObject method
  - Invoked for any object defined in your ADE to be exported from the database
  - You must **only export ADE-specific properties** defined for the object
  - Non-ADE properties (i.e., properties inherited from CityGML object types) **are managed by the Importer/Exporter**
  - There are no restrictions on the way you use JDBC

- Input
  - ADEModelObject object: the citygml4j object as defined in your citygml4j ADE module
  - long objectId: the assigned database ID
  - AbstractObjectType<?> objectType: the object type from the schema mapping file
  - ProjectionFilter filter: a filter defining which attributes shall be exported or skipped for the given ADE object

ADEExportManager skeleton (3/4)

- exportGenericApplicationProperties method
  - Invoked for „ADE hook properties“ defined in your ADE and found in the database
  - All ADE properties stored in one hook table must be processed in one call
  - You must **only export the ADE hook properties** (not the parent feature)
  - There are no restrictions on the way you use JDBC

- Input
  - String adeHookTable: The table name from which the ADE properties shall be exported
  - AbstractFeature parent: The feature to which the ADE properties must be assigned
  - long parentId: The database ID of the feature
  - FeatureType parentType: the feature type from the schema mapping file
  - ProjectionFilter filter: a filter defining which attributes shall be exported or skipped for the given ADE object

ADEExportManager skeleton (4/4)

- close method
  - Close all database (and possibly further) resources that you have opened in your own exporter classes
  - Do not close resources passed to your classes; they are managed by the Importer/Exporter
  - Invoked at the end of the export session

Batch export

- **NOTE:** The Importer/Exporter uses batch processing when exporting objects from the 3DCityDB
  - To reduce the number of SQL queries for exporting an object
  - To increase the export performance
- When ADE objects are passed to the methods of the ADEExportManager interface, they do not yet contain the data from the core CITYOBJECT table
- If you need a specific information from the CITYOBJECT table for processing your ADE object like the gml:id, then either
  - explicitly query this information by joining the table (recommended), or
  - invoke the method executeBatch of the CityGMLExportHelper to immediately populate the corresponding object properties (at the cost of performance)

Export utility classes making your life easier

CityGMLExportHelper utility (1/7)

```
public interface CityGMLExportHelper {
   <T extends AbstractGML> T createObject(long objectId, int objectClassId, Class<T> type) throws CityGMLExportException, SQLException;
   <T extends AbstractFeature> Collection<T> exportNestedCityGMLObjects(FeatureProperty featureProperty, long parentId, Class<T> featureClass) throws CityGMLExportException, SQLException;
   ImplicitGeometry createImplicitGeometry(long id, GeometryObject referencePoint, String transformationMatrix) throws CityGMLExportException, SQLException;
   SurfaceGeometryExporter getSurfaceGeometryExporter() throws CityGMLExportException, SQLException;
   AttributeValueSplitter getAttributeValueSplitter();
   GMLConverter getGMLConverter();

   void executeBatch() throws CityGMLExportException, SQLException;
   boolean exportAsGlobalFeature(AbstractFeature feature) throws CityGMLExportException, SQLException;
   boolean supportsExportOfGlobalFeatures();

   AbstractDatabaseAdapter getDatabaseAdapter();
   CityGMLVersion getTargetCityGMLVersion();
   ProjectionFilter getProjectionFilter(AbstractObjectType<?> objectType);
   CombinedProjectionFilter getCombinedProjectionFilter(String tableName);
   LodFilter getLodFilter();

   boolean isFailOnError();
   ExportConfig getExportConfig();

   String getTableNameWithSchema(String tableName);
   ProjectionToken getGeometryColumn(Column column);
   ProjectionToken getGeometryColumn(Column column, String asName);
   String getGeometryColumn(String columnName);
   String getGeometryColumn(String columnName, String asName);

   void logOrThrowErrorMessage(String message) throws CityGMLExportException;
   String getObjectSignature(int objectClassId, long id);
   String getObjectSignature(AbstractObjectType<?> objectType, long id);

   FeatureType getFeatureType(AbstractFeature feature);
   ObjectType getObjectType(AbstractGML object);
   AbstractObjectType<?> getAbstractObjectType(AbstractGML object);
   FeatureType getFeatureType(int objectClassId);
   ObjectType getObjectType(int objectClassId);
   AbstractObjectType<?> getAbstractObjectType(int objectClassId);

   boolean lookupAndPutObjectId(String gmlId, long id, int objectClassId);
   boolean lookupObjectId(String gmlId);
}
```

© VC Systems

CityGMLExportHelper utility (2/7)

```
<T extends AbstractGML> T createObject(long objectId,
                           int objectClassId, Class<T> type);
<T extends AbstractFeature> Collection<T> exportNestedCityGMLObjects
                  FeatureProperty featureProperty, long parentId,
                  Class<T> featureClass);
ImplicitGeometry createImplicitGeometry(long id, GeometryObject
                 referencePoint, String transformationMatrix);
SurfaceGeometryExporter getSurfaceGeometryExporter();
AttributeValueSplitter getAttributeValueSplitter();
GMLConverter getGMLConverter();
```

- createObject method
  - Used to create a GML object for a given database ID and object class ID
  - The Importer/Exporter takes care of the CityGML-specific properties
  - **You have to export the ADE-specific properties**

- exportNestedCityGMLObjects method
  - Use this method if your ADE object has a feature property that points to a CityGML feature (or a subtype thereof)
  - It will export and return all nested features as a collection
  - If a nested feature is itself an ADE object, the Importer/Exporter **will call back** your ADEExportManager

CityGMLExportHelper utility (3/7)

```
<T extends AbstractGML> T createObject(long objectId,
                           int objectClassId, Class<T> type);
<T extends AbstractFeature> Collection<T> exportNestedCityGMLObjects
                  FeatureProperty featureProperty, long parentId,
                  Class<T> featureClass);
ImplicitGeometry createImplicitGeometry(long id, GeometryObject
                 referencePoint, String transformationMatrix);
SurfaceGeometryExporter getSurfaceGeometryExporter();
AttributeValueSplitter getAttributeValueSplitter();
GMLConverter getGMLConverter();
```

- createImplicitGeometry method
  - Used to create an ImplicitGeometry object from a reference point, a transformation matrix and a primary key ID of IMPLICIT_GEOMETRY
- AttributeValueSplitter
  - Counterpart of AttributeValueJoiner
- SurfaceGeometryExporter
  - Used to export a geometry from SURFACE_GEOMETRY via its primary key ID
  - Due to batch processing, **the geometry is not returned directly** but you have to provide a **geometry setter method** (functional interface)
  - If you need the geometry immediately, invoke executeBatch
- GMLConverter
  - Used to convert a GeometryObject into a citygml4j geometry

CityGMLExportHelper utility (4/7)

```
void executeBatch();
boolean exportAsGlobalFeature(AbstractFeature feature);
boolean supportsExportOfGlobalFeatures();

AbstractDatabaseAdapter getDatabaseAdapter();
CityGMLVersion getTargetCityGMLVersion();
ProjectionFilter getProjectionFilter(AbstractObjectType<?> objectType);
CombinedProjectionFilter getCombinedProjectionFilter(String tableName);
LodFilter getLodFilter();
```

- executeBatch method
  - Execute the current batch to get access to CITYOBJECT properties and SURFACE_GEOMETRY objects in your ADE object
  - **At the cost of performance**
- getLodFilter method
  - Get the LodFilter used for the export
- exportAsGlobalFeature method
  - Export a feature as <gml:featureMember> of the root <CityModel> if supported by the writer
- getProjectionFilter method
  - Get projection filter for a given object type
- getCombinedProjectionFilter method
  - Get projection filter for all object types stored in the same table

CityGMLExportHelper utility (5/7)

```
String getTableNameWithSchema(String tableName);
ProjectionToken getGeometryColumn(Column column);
ProjectionToken getGeometryColumn(Column column, String asName);
String getGeometryColumn(String columnName);
String getGeometryColumn(String columnName, String asName);

void logOrThrowErrorMessage(String message) throws CityGMLExportException;
String getObjectSignature(int objectClassId, long id);
String getObjectSignature(AbstractObjectType<?> objectType, long id);
```

- getTableNameWithSchema method
  - Adds the schema name of the database connection as prefix to your table name (“schema.tableName”)

- getGeometryColumn methods
  - Gets a geometry column name for use in an SQL statement or with the sqlbuilder library
  - Adds spatial functions to apply coordinate transformations (if required by the export)

- Logging helpers
  - Some helper methods to create standardized log messages
  - getObjectSignature method turns an object into a String representation that can be used in log messages

CityGMLExportHelper utility (6/7)

```
ExportConfig getExportConfig();

FeatureType getFeatureType(AbstractFeature feature);
ObjectType getObjectType(AbstractGML object);
AbstractObjectType<?> getAbstractObjectType(AbstractGML object);
FeatureType getFeatureType(int objectClassId);
ObjectType getObjectType(int objectClassId);
AbstractObjectType<?> getAbstractObjectType(int objectClassId);
```

- ExportConfig
  - Get access to all export preference settings made by the user for the current export session
- SchemaMapping helpers
  - Some helper methods to get the object type definition from the schema mapping for a given object class ID or citygml4j object

CityGMLExportHelper utility (7/7)

```
boolean lookupAndPutObjectId(String gmlId, long id, int objectClassId);
boolean lookupObjectId(String gmlId);
```

- lookupAndPutObjectId method
  - Returns true if a given object identifier (gml:id) has been previoulsy put into the local cache; in this case you will most likely create an XLink in your code
  - Returns false if the object identifier is not available from the local cache; in this case it will be put into the local cache so that subsequent checks will return true
- lookupObjectId method
  - Just check whether a given object identifier (gml:id) has been previoulsy put into the local cache **without actually putting it into the cache**
  - Again, is true is returned, you will most likely create an XLink in your code

Further interfaces and utilities (1/2)

- DatabaseAdapter
  - Used to get **metadata** about the currently connected database and to **execute operations** on this database
  - See description in import part of this presentation

Most important functionality when exporting ADE data:

- GeometryConverter
  - Used to convert a database-specific geometry object stored in a geometry column into a database-agnostic instance of GeometryObject
  - The GeometryObject can afterwards be converted into a citygml4j object using the GMLConverter from CityGMLExportHelper

Further interfaces and utilities (2/2)

- ADEExporter interface
  - Simple interface that you can use for the export classes working on different citygml4j objects and tables
  - Optional but helpful

Implement the ADEBalloonManager interface to export ADE attribute data

ADEBalloonManager skeleton (1/4)

```
public class BalloonManager implements ADEBalloonManager {

   @Override
   public ADEBalloonHandler getBalloonHandler(int objectClassId) throws ADEBalloonException {
      return new ADEBalloonHandler() { … };
   }

   @Override
   public HashMap<String, Set<String>> getTablesAndColumns() {
      return new HashMap<>();
   }

}
```

ADEBalloonManager skeleton (2/4)

- getBalloonHandler method
  - Invoked for any top-level features defined in your ADE
  - Returns an instance of a handler class, which shall implement the interface ADEBalloonHandler
  - Each handler class should be defined for to a specific ADE top-level feature class for handling the attribute data export
  - Input
    - objectClassId: the object class ID representing the corresponding object type from the schema mapping file
- getTablesAndColumns method
  - Returns a HashTable for representing the ADE database tables and their columns.
    - Each key of the HashTable should be an ADE table name and upper-case
    - The value of each key should be a String set, which lists the names of the corresponding attribute columns

ADEBalloonManager skeleton (3/4)

```
public class BuildingBalloonHandler implements ADEBalloonHandler {

   @Override
   public String getSqlStatement(String table,
                                 String tableShortId,
                                 String aggregateColumnsClause,
                                 int lod,
                                 String schemaName) {

      return "";
   }

}
```

ADEBalloonManager skeleton (4/4)

- getSqlStatement method
  - Invoked for retruning an SQL statement for exporting attributes from different tables related to an ADE top-level feature class

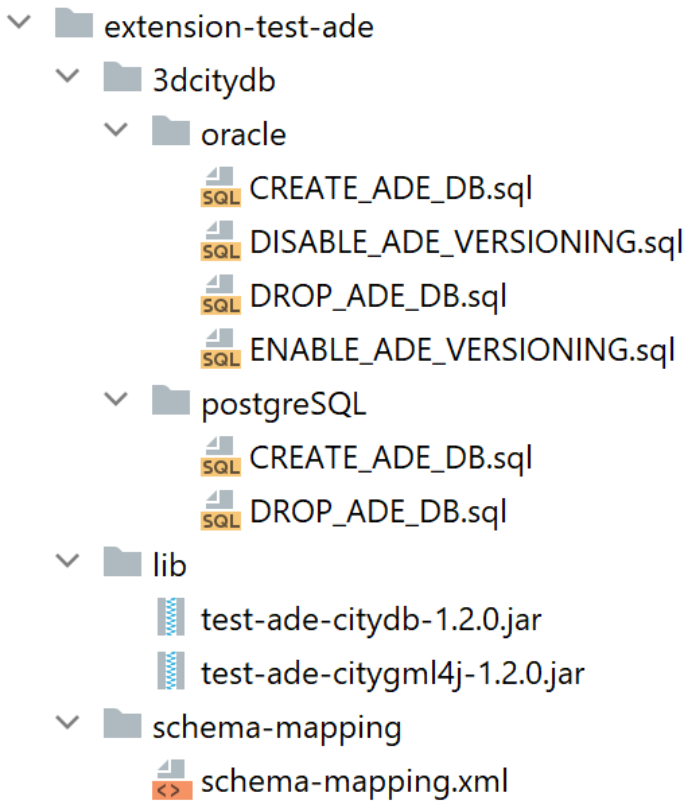
- Input
  - String table: The name of the table from which the attribute data will be exported
  - String tableShortId: The shorthand (alias) of the corresonding table
  - String aggregateColumnsClause: column name prefixed with the table alias. It could also be wrapped with an SQL function e.g. min, max, count
  - int lod: You could use this parameter to export different attributes based on different LODs.
  - String schemaName: The name of the target database schema

Create a distribution package for your ADE extension

Package your ADE extension

- All ADE extension packages **MUST** follow a predefined structure

- 3dcitydb folder
  - Contains the SQL scripts to set up the ADE schema in the database for Oracle and PostgreSQL
- schema-mapping folder
  - Contains your schema-mapping file
- lib folder
  - Contains your implementation of the ADEExtension interface plus supporting classes/libs (e.g. your citygml4j ADE module)



Super easy with Gradle

```
plugins {
    id 'java-library'
    id 'distribution'
}

group 'org.citydb.ade'
version '1.0-SNAPSHOT'

configurations {
    citygml4j
}


dependencies {
    api 'org.citydb:impexp-client:4.3.0'
    implementation 'org.citygml4j.ade:hft-test-ade-citygml4j:1.0.0'

    citygml4j('org.citygml4j.ade:hft-test-ade-citygml4j:1.0.0') {
        transitive = false
    }
}

distributions.main {
    distributionBaseName = 'extension-test-ade'

    contents {
        into('lib') {
            from jar
            from configurations.citygml4j
        }
        into('schema-mapping') {
            from "$rootDir/resources/schema-mapping"
        }
        into('3dcitydb') {
            from "$rootDir/resources/3dcitydb"
        }
    }
}
```

- Use the distribution plugin
- The package contents are defined in the distribution.main section
- For the citygml4j module, we can skip the transitive dependencies of the citygml4j library (transitive = false)
  - They are already loaded by the Importer/Exporter

Using your ADE extension package with the Importer/Exporter

- Simply copy your ADE extension package into the ade-extensions folder within the installation directory of the Importer/Exporter
- Your ADE extension should be automatically loaded when (re-)starting the Importer/Exporter
- More information available at: https://3dcitydb-docs.readthedocs.io/en/version-2021.1/plugins/ade-manager/impexp-ade-extension.html

Thank you
for your attention

www.vc.systems