**1. Description of the parser module**

In the following chapter of the documentation, we will describe the function and the usage of the parser module. This module covers the following functionality of the vCity software:

* Import and parsing of the CityGML file
* Vertex translation to a defined reference coordinate
* Polygon tessellation
* Three different export variations

**7. Data Export**

**CSV**

This option will write the building IDs and their volumes to a simple CSV (**C**omma **S**eparated **V**alues) file.

For this export method, we decided to use a simple java **FileWriter**, as using a framework seemed to be unnecessary to us.

Building,Volume

DEBW\_LOD2\_1007722,112.82454681396484

**GML**

Using this option, the volume of each building will be attached as a double value to the imported city model and will be exported as a new GML (**G**eography **M**arkup **L**anguage) file.

The GML file is written using the **CityGML4J** framework, which is also used for the import. The framework is released under the GNU LGPL license. We use the imported **CityModel** object and add a double value to the buildings. The object is parsed back and written out to the destination file.

<gen:doubleAttribute name="Volume">

<gen:value>112.82454681396484</gen:value>

</gen:doubleAttribute>

**XML**

This export method is designed to write out a XML (*e****X****tensible* ***M****arkup* ***L****anguage*) file that can be used in the “**INSEL**” software. The XML data formatting is designed to fit the requirements that have been evaluated in a customer review meeting. Please note that the GML Polygons and other attributes are not saved in this file format.

The DTD (**D**ocument **T**ype **D**efinition) of the XML file is defined as following:

<!ELEMENT City (SkyModel, Building+)>

<!ELEMENT SkyModel (SplitAzimuth, SplitHeight)>

<!ELEMENT SplitAzimuth (#PCDATA)>

<!ELEMENT SplitHeight (#PCDATA)>

<!ELEMENT Building (BoundarySurface)+>

<!ATTLIST Building

id CDATA #REQUIRED

>

<!ELEMENT Volume (#PCDATA)>

<!ATTLIST Volume

uom CDATA #REQUIRED

>

<!ELEMENT BoundarySurface (Polygon)+>

<!ATTLIST BoundarySurface

id CDATA #REQUIRED

type (GROUND|ROOF|WALL|OTHER) #REQUIRED

>

<!ELEMENT Polygon (Area, Shadow)>

<!ATTLIST Polygon

id CDATA #REQUIRED

>

<!ELEMENT Area (#PCDATA)>

<!ATTLIST Area

uom CDATA #REQUIRED

>

<!ELEMENT Shadow (#PCDATA)>

We use the **StAX** library, released under the Apache v2.0 license, to create the XML data object. We decided to use this library because its performance is good and it is simple to use.

We create the XML objects using Element e = doc.createElement("Element"); . Attribute values are created using Attr a = doc.createAttribute("Attribute"); . These are very useful, because, to create and fill the XML object, we are able to simply walk over each building in our model and read the data needed.

for (each Building)

add ID;

add Volume;

for (each Surface)

...

**8. Test Cases**

To test the functionality of the Import and Export, we wrote a JUnit test class. The class contains various test cases for import, parsing, tessellation and translation. The actual biggest test case is testReadAndParseAndValidateEinHaus(), which contains a full parsing and data validation of the Gruenbuehl GML file from the very top, down to the [x,y,z] coordinates of a Vertex. The three export variations are performed too, but not fully tested, meaning the result files need to be checked manually.