BIM 5D Planning – Query code outline and description

1. Setting up the connection to the BIM server

- open the file/class “QueryMain.java”
- the code at the top of the class looks like this:

```
public class QueryMain {

        private static String name = "l.l.oldescholtenhuis@utwente.nl";
        private static String password = "password";
        private static String server = "http://bim.utwente.nl:8080";
```

- to connect to your BIM database, please change the name and password attribute to your own credentials (name: email address as on blackboard, password).
- in the file/class “ExportQuantities.java” at the beginning of the file code looks like this:

```
        // this uses the example project name, please exchange it with the name of your group's project
        private static final String projectName = "Test3";
```

- change the projectName string to the name of your project on the BIM server (your first name)
- now you should be able to test run the three demonstration functions “queryAll”, “queryAllByStorey”, and “exportWallAreaPerPhase” in the public static void main() in the “ExportQuantities.java” file at the top:

```
public class ExportQuantities
{
…
        public static void main(String[] args) throws ServerException, UserException, RowsExceededException,
        BiffException, WriteException, IOException, BimServerClientException,
        PublicInterfaceNotFoundException
        {
        …
                //queryAll();
                //queryAllByStorey();
                exportWallAreaPerPhase();
        }
```

- The characters // are “commenting a line out” - which means lines marked with // are not considered during the execution of the program. If there are no “//” in the code, then all lines will be executed. To test each of the functions separately, remove the “//” before specific function name (if any) and set the “//” before the other two.

The rest of this document describes each of the three demo functions in detail.

2. queryAll() demo function

The queryAll() function is the simplest example of how to query specific IFC based objects from the BIM server and print their properties to the console. Once understood this function serves as the basic building block for all BIM server related queries. This function consists of three parts:

- establish a connection to the BIM server and to the latest revision of your specific personal project on the server.

```
QueryMain m = new QueryMain();
ClientIfcModel model = m.getModel(projectName);
```

- This part of the function establishes a new QueryMain object and gets the model earlier defined as a variable “projectName”.

- collecting all objects of type door, column, roof, stair, window, slab, standardwallcase

```
List<IfcObject> objects = new ArrayList<IfcObject>();

objects.addAll(model.getAllWithSubTypes(IfcWallStandardCase.class));
objects.addAll(model.getAllWithSubTypes(IfcColumn.class));
objects.addAll(model.getAllWithSubTypes(IfcRoof.class));
objects.addAll(model.getAllWithSubTypes(IfcStair.class));
objects.addAll(model.getAllWithSubTypes(IfcWindow.class));
objects.addAll(model.getAllWithSubTypes(IfcSlab.class));
objects.addAll(model.getAllWithSubTypes(IfcColumn.class));
objects.addAll(model.getAllWithSubTypes(IfcWall.class));
objects.addAll(model.getAllWithSubTypes(IfcDoor.class));
```

First we establish a list (‘objects’) that allows us to store the different building elements. We then use it select all objects of a certain ifc type from the BIM server. Please refer to the ifc documentation and use the browse function of the BIM server web interface to understand the existing objects in your specific project database.

– iterate the objects list and print each objects' properties

```
Iterator<IfcObject> objectIt = objects.iterator(); (1)

while (objectIt.hasNext()) (2)
{
	IfcObject object = objectIt.next(); (3)
	PropertyObject qo = new PropertyObject(object); (4)
	qo.printProperties(); (5)
}
```

The above code finally prints the properties to the console (in eclipse). Please note first the basic java functionality to iterate over each object within a list (following the numbering scheme in the above code fragment):

1. we obtain an iterator for the list (please watch the type of the iterator given by the brackets “<>” - in this case IfcObject) using the lists function “.iterator()” (tutorials that describe the use of lists, arrays, and iterators in JAVA can be found online).

2. Using the iterator we can ask whether there is a next element with the “hasNext()” function. Please note that the while command enters a loop that will be repeated until the iterator has no next element (more information about JAVA loops can be found in the eBook that is referenced on blackboard).
3. With the “next()” function of the iterator we can then get the next object.
4. We establish an object PropertyObject that takes all referenced property sets of the specific objects from the local container.
5. The function “qo.printProperties()” prints these properties to the console.
6. For more information about the existing PropertySets please refer to the IFC documentation and browse your specific project using the BIM Server web interface.

Please note the function of the PropertyObject class. It extracts all IFCPropertySets that are connected to a specific object (using its ObjectContainer class). These properties can then be queried and accessed locally:

- the printProperties() functions prints all of them to the console,
- the exportAllProperties() function exports all properties to an Excel spreadsheet
- the exportPropertyToExcel() function exports one specific property to a specific cell in a Excel spreadsheet
- the extractValue() function returns the value of a specific property as a string object

A demonstration of some of the functionality is illustrated in the provided test function exportWallAreaPerPhase(); described in the next section.

3. queryAllByStorey() function

The queryAllByStorey function is an example of how IFC grouping objects such as storeys can be used to get aggregated information about objects. This is also a great example to see how the all connected objects are pulled from the server to the local computer. Please again refer to the IFC standard and use the browse functionality of the BIM server web interface to understand how the objects are related to each other.

Here is the code and its description:

```
QueryMain m = new QueryMain();
ClientIfcModel model = m.getModel(projectName);


List<IfcBuildingStorey> objects = model.getAllWithSubTypes(IfcBuildingStorey.class);

for (IfcBuildingStorey storey : objects)
{
        System.out.println();
        System.out.println();
        System.out.println(storey.getName());                   (1)
        List<IfcRelContainedInSpatialStructure> contained = 
storey.getContainsElements();
        for (IfcRelContainedInSpatialStructure c : contained)
        {
                for (IfcObject el : c.getRelatedElements())       (2)_
                {
                        PropertyObject qo = new PropertyObject(el);
                        qo.printProperties();
                }
        }
}
```

We again start with establishing a QueryMain object and connect to the server. Similar to the first example, we then obtain all elements of a specific type – in this case the IfcBuildingStorey – object from the server. Within the for loop we first print all the names of the storey object (1). Then we establish a list named “contained” that consists of elements “IfcRelContainedInSpatialStructure”. Then we go through the list taking one record of this list one after another: line “for (IfcRelContainedInSpatialStructure c : contained)”. Each of the records in the list “contained” is constituted by several elements. We go through the elements using the “for” loop (2) made using “`getRelatedElements()`” function.
. In the loop we print all properties of the connected objects.

4. <u>exportWallAreaPerPhase()</u>

The exportWallAreaPerPhase function is a more specialized example. It shows how a specific quantity stored as a property of the IFCWallStandardCase object can be aggregated for all objects on a specific storey according to another property of the IFCWallStandardCase object – “Phase Created” to an Excel spreadsheet. This function serves as an example of how to meaningfully export aggregated quantity information from projects stored on the BIM server to Excel cost estimation sheets.

– The function starts with establishing a connection to the BIM server.

```
public static void exportWallAreaPerPhase() throws ServerException, UserException, BiffException,
IOException, RowsExceededException, WriteException

{
        QueryMain m = new QueryMain();
        ClientIfcModel model = m.getModel(projectName);List<IfcBuildingStorey> objects =
model.getAllWithSubTypes(IfcBuildingStorey.class);

        // get the first storey object
        IfcBuildingStorey storey = objects.get(0);
```

The connection to the server is similar to the above two examples. We connect to the BIM server and get all storey objects. Then we select the first storey object. If your model has multiple storey objects, you can use the BIM server’s browser functionality to identify the right number. Please be aware that JAVA starts counting at 0, so the first storey object shown by the BIM server will be the object with the number 0.

As we will need an Excel spreadsheet please see how we defined it earlier in the file.

The connection to Excel is defined earlier, in the class function ‘main()’ in the following form:

```
sheet = new ExcelSheet("demo/phases.xls", "demo/phases_new.xls", "Quantities");
…
sheet.writeAndClose();
```

Please note the syntax of the ExcelSheet constructor function: an existing spreadsheet (demo/phases.xls) is taken and copied to a new sheet (demo/phases_new.xls) for writing information to. In this way, sheets with a pre-existing structure can be used as a template (here “phases.xls”). The “Quantities” string signifies to the program to which sheet in a xls spreadsheet the information should be written. Please refer to the structure of the with the zip file delivered Excel spreadsheet. This information can be adjusted in your own query code according to your specific preferences.

```
// create the map to hold the length per phase
HashMap<String, Double> lengthPerPhase = new HashMap<String, Double>();
```

The above code also establishes another container. Other than the other two containers List (just a list of objects) and Set (a set of unique obejcts), the container HashMap allows us to store information – in this case double numerical values - related to a specific index – in this case a String. This will allow us to store the length of walls that exists for each construction stage and later write it to the Excel spreadsheet.

- the next part of the function that iterates over all objects and fills the HashMap with information.

```
List<IfcRelContainedInSpatialStructure> contained = storey.getContainsElements();
for (IfcRelContainedInSpatialStructure c : contained)
{
        for (IfcObject el : c.getRelatedElements())
        {
                // select only walls
                if (el instanceof IfcWallStandardCase || el instanceof IfcWall)
                {
PropertyObject qo = new PropertyObject(el);


String phase = qo.getQuantity("Phase Created");
String lengthStr = qo.getQuantity("Length");
```

To select only wall object we use the function getContainedStrcuturesByType (1) that returns all walls of the respective story object. We then iterate over each of these walls and extract the value of the property "PhaseCreated" and "Length" using the PropertyObject class that we instantiate using each wall object. To see a complete list of properties of an object please use the code from the two above functions and browse the project using the web-interface of the IFC server. This function shows how to receive specific properties of an object. These properties are always returned as String objects. To make meaningful calculations, these Strings need to be converted to numerical objects such as Double as shown in the above example.

The next step than aggregates the length of the walls per phase using the initially established HashMap object:

```
// no property set, ignore wall
if (phase != null)
{
        // already a wall with this phase: add length to value
        if (lengthPerPhase.containsKey(phase))
        {
                Double aggregatedLength = lengthPerPhase.get(phase);
                aggregatedLength = aggregatedLength + length;
                lengthPerPhase.put(phase, aggregatedLength);
        }
        // create new HashMap entry
        else
        {
                lengthPerPhase.put(phase, length);
        }
```

First, this function checks whether a phase object exists for this specific IFCWallStandardCase. This check is important for not crashing the program in case there is no single IFCWallStandardCase object. Then we check whether the HashMap already contains an entry with the key of the phase name of the object: if (lengthPerPhase.containsKey(phase)). If yes, we add the length of the new wall object to the aggregated length of all previous objects with this phase. Otherwise (else), we add a new phase entry to the Map with the initial length of this object as a starting value. We then do this procedure for each of the IFCWallStandardCase objects.

Finally, we write the information to the Excel spreadsheet with the following code:

```
// write HashMap to Excel

Iterator<String> keyIt = lengthPerPhase.keySet().iterator();

//start at row 2 to spare heading
int row = 2;

while (keyIt.hasNext())
{
        String phase = keyIt.next();
        Label labelCell = new Label(1, row, phase);
        sheet.getSheet().addCell(labelCell);

        Number number = new Number(2, row, lengthPerPhase.get(phase).doubleValue())
        sheet.getSheet().addCell(number);
        row++;
}
```

Again we use the iterator functionality, this time to iterate over each entry in the HashMap. For each entry we write the phase name into the Excel spreadsheet in the first column starting with row two for the first entry. In the second row we then add the length value for this phase. We then add one to the row (row++) and continue the loop with the next phase entry in the HashMap. Please be aware that the row value is increased by one every time the loop is run through – this ensures that each aggregated length pair for each of the phases is written to a separate cell. Please refer to the Excel spreadsheet created to see the final output.