**Working with parameters in Revit API**

1. **Using Lookup Parameter**

In this method you can retrieve any element parameter by selecting the element and then retrieve the parameter by the parameter’s name .

1. // Select Element First
2. var element = uiDocument.Selection.GetElementIds().Select(x => doc.GetElement(x)).First();
3. // Retrieving any parameter by its name (ex. Area)
4. var value = element.LookupParameter("Area").AsValueString();
5. **Using BuiltInParameter**

In this method you can retrieve any element parameter by selecting the element and then retrieve the parameter by the BuiltInParamter and this method is better than using Lookup Parameter.

1. // Retriving any parameter of the element by using BuiltInParamter (ex. Area)
2. var value = element.get\_Parameter(BuiltInParameter.HOST\_AREA\_COMPUTED).AsValueString();
3. **Adding value to Paramters of an element**

In this topic you can add values to parameters of the selected element .

1. // Adding Paramter to selected element (ex. Adding “Str Walls” to Comments Parameter)
2. using (Transaction tr = new Transaction(doc, "Set Value Paramter"))
3. {
4. tr.Start();
5. // Adding value to comments Parameter (ex. Str Walls to selected Wall)
6. element.get\_Parameter(BuiltInParameter.ALL\_MODEL\_INSTANCE\_COMMENTS).Set("Str Walls");
8. tr.Commit();
9. }
10. **Retrieving Parameters data type**

In this topic you can know the parameter data type (String, Integer, ElementId, Double,.. etc)

Using *LookUpParameter* to retrieving the parameters that user want to examine

1. string storageType = string.Empty; // == string storageType = “”;
3. Parameter comment = ele.LookupParamter(“Comment”);
4. Parameter topOffset = ele.LookupParamter(“Top Offset”);
5. Parameter MovewithGrid = ele.LookupParamter(“Move With Grids”);
6. Parameter topLevel = ele.LookupParamter(“Top Level”);
7. Parameter woodMaterial = ele.LookupParamter(“Structural Material”);
8. storageType = “The Storage type of the following parameters is :” +
9. Environment.NewLine + Environment.NewLine // new Line
10. + “Comment Pararmeter is :” + comment.StorageType.ToString()
11. + Environment.NewLine + “Top Offset parameter is :”
12. + topOffset.StorageType.ToString();
13. TaskDialog.Show(“Parameter Storage type”, storageType);

**Selection Filter in Revit API**

In this topic you can use *ISelectionFilter* to filtering the category or the specific elements you want to make operations on it .

Ex. Selection filter for Walls only

1. Make new class named it “*ElementSelectionFilter .cs*“
2. Write the following
3. public class ElementSelectionFilter : ISelectionFilter
4. {
5. bool ISelectionFilter.AllowElement(Element elem)
6. {
7. // Selected Walls Only
8. return (BuiltInCategory)GetCategoryIdAsInteger(elem) == BuiltInCategory.OST\_Walls;
9. }
11. bool ISelectionFilter.AllowReference(Reference reference, XYZ position)
12. {
13. return false;
14. }
16. public int GetCategoryIdAsInteger(Element element)
17. {
18. return element?.Category?.Id?.IntegerValue ?? -1;
19. }
20. In the main class use selection filter as following
21. var reference = uiDocument.Selection.PickObjects(ObjectType.Element, new ElementSelectionFilter());
22. in “*ElementSelectionFilter .cs*“ class if you want to change wall to any other category just change line number 6
23. // Selected Floors Only
24. return (BuiltInCategory)GetCategoryIdAsInteger(elem) == BuiltInCategory.OST\_Floors;

Another way of filtering elements in Revit is using “*FilteredElementCollector* ”

Ex. Collect all columns type in the active document and store it in a list

1. // Filtering Elements using BuiltInCategory
2. IList<Element> columns = new FilteredElementCollector(doc)
3. .OfCategory(BuiltInCategory.OST\_StructuralColumns)
4. .WhereElementIsNotElementType().ToElements();
6. IList<string> columnTypes = new List<string>();
8. // Iterating throw the columns collection
9. foreach (Element ele in columns)
10. {
11. // adding the columns types in the list
12. columnTypes.Add(ele.Name);
13. }

Another way to filtering elements in Revit is to filtering **All Family Instances by Name** so the logic has to be as the following sequence:

* 1. Filter for Family
  2. Determine their types
  3. Then the instances of this family type

1. String familyName = “Family Name”;
2. String familyTypeName = “Family Type Name”;
3. // get all instances by family name then type name
4. IList<FamilyInstance> famInstances = new FilteredElementCollector(doc)
5. . OfClass (typeof (FamilyInstance))
6. . Cast<FamilyInstance> ()
7. . Where(x=> x.Symbol.Family.Name.Equals(familyName) //Family name
8. . Where(x=> x. Name. Equals(familyTypeName)) //Family Type name
9. . ToList<FamilyInstance> ();