Revit API Intro Labs

Lab2 – DB Element

Created by M. Harada, July 2010
Updated by DevTech AEC WG
Last modified: 10/8/2023

<C#>C# Version**</C#>**

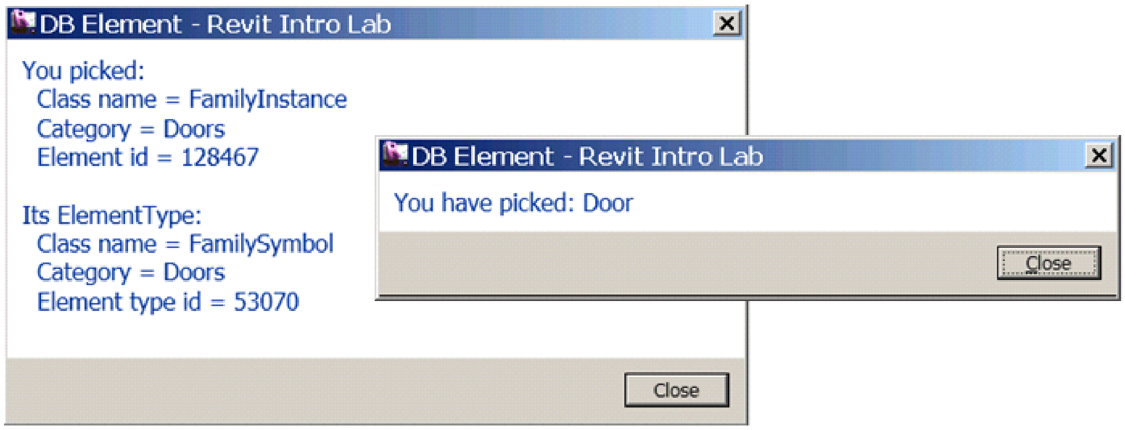
Objective: In this lab, we will learn how an element is represented in Revit and how to retrieve information about an element. We'll learn how to:

- Identify an element
- Retrieve a set of properties of an element
- Retrieve a specific property of an element
- Retrieve location information
- Retrieve geometry information

Tasks: We'll write a command that prompts the user to pick an element, identify the kind of element, and displays its information about properties, location and geometry:

1. Pick an element.
2. Show basic information about the element and its family type (i.e., class name, category and element Id).
3. Identify the element that you have picked. (e.g., is it a wall, window or door?)
4. Show the set of properties of the element and its family type.
5. Show specific properties of the element and its family type.
6. Show the location information of the element
7. Show the geometry information of the element (Optional)

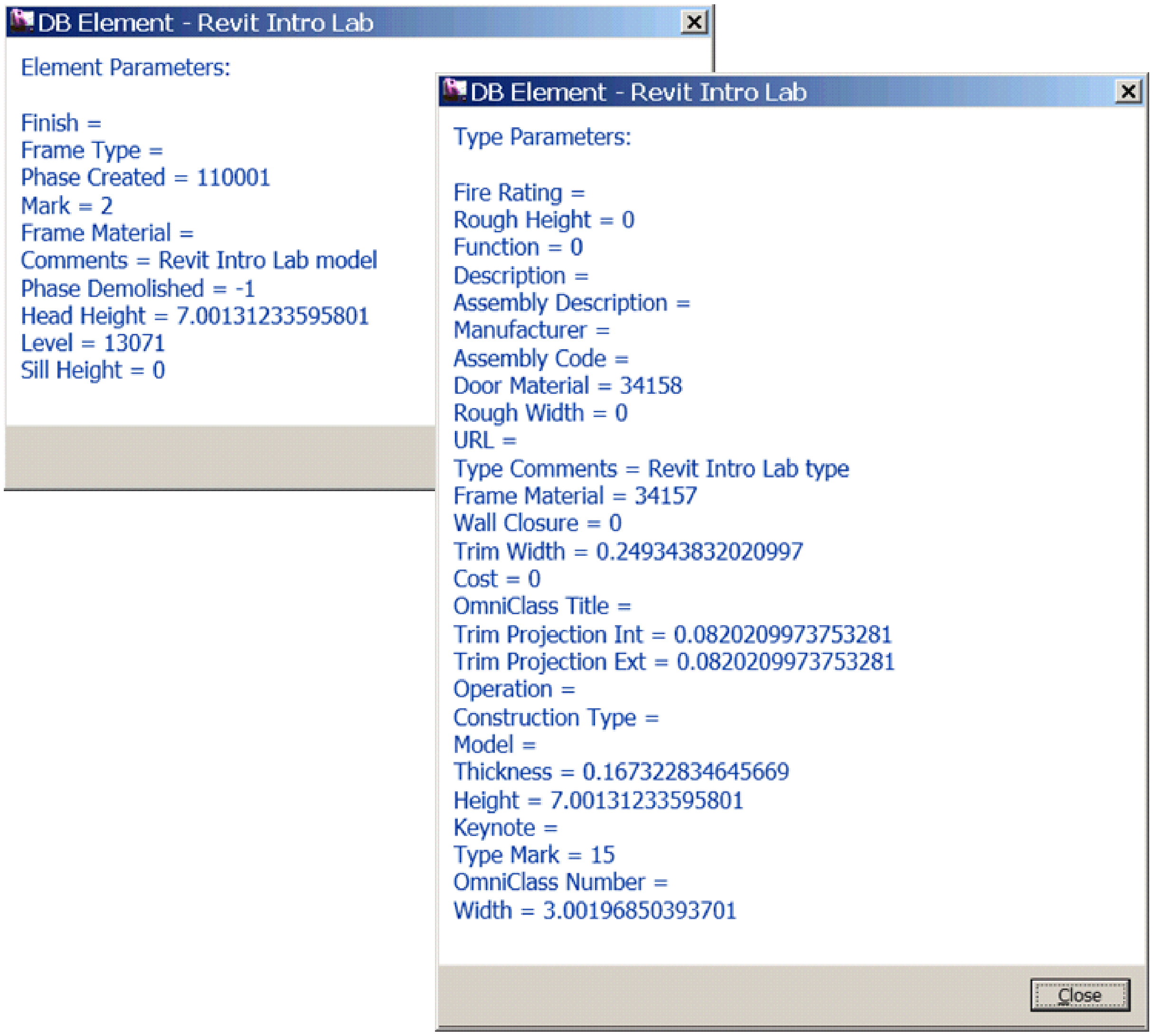
Figure 1 and 2 shows the sample images of output after running the command that you will be defining in this lab:



```
You picked:
  Class name = FamilyInstance
  Category = Doors
  Element id = 128467

Its ElementType:
  Class name = FamilySymbol
  Category = Doors
  Element type id = 53070
```

```
You have picked: Door
```

Figure 1. Dialogs showing basic information and identity of an element.



```
Element Parameters:

Finish =
Frame Type =
Phase Created = 110001
Mark = 2
Frame Material =
Comments = Revit Intro Lab model
Phase Demolished = -1
Head Height = 7.00131233595801
Level = 13071
Sill Height = 0
```

```
Type Parameters:

Fire Rating =
Rough Height = 0
Function = 0
Description =
Assembly Description =
Manufacturer =
Assembly Code =
Door Material = 34158
Rough Width = 0
URL =
Type Comments = Revit Intro Lab type
Frame Material = 34157
Wall Closure = 0
Trim Width = 0.249343832020997
Cost = 0
OmniClass Title =
Trim Projection Int = 0.0820209973753281
Trim Projection Ext = 0.0820209973753281
Operation =
Construction Type =
Model =
Thickness = 0.167322834645669
Height = 7.00131233595801
Keynote =
Type Mark = 15
OmniClass Number =
Width = 3.00196850393701
```

Figure 2. Dialogs showing properties (or parameters) of an element and of its type

The following is the breakdown of step by step instructions in this lab:

1. Define A New External Command
2. Pick an Element
3. Basic Element Information
4. Identify Element
5. Parameters
6. Location Information
7. Geometry Information (Optional)
8. Summary

1. Define A New External Command

We'll add another external command to the current project. Do not create a new project, but use the existing one.

1.1 Add a new file and define another external command to your project. Let's name them as follows:

- File name: **2_DbElement.vb (or .cs)**
- Command class name: **DBElement**

(Once again, you may choose to use any names you want here. When you do so, just remember what you are calling your own project, and substitute these names as needed while following the instruction in this document.)

Required Namespaces:
Namespaces needed for this lab are:

- (System.Linq) (needed for Lab3)
- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.ApplicationServices
- Autodesk.Revit.Attributes
- Autodesk.Revit.UI.Selection (this is for selection)

Note (VB.NET only): if you are writing in VB.NET and you import namespaces at the project level, (i.e., in the project properties, there is no need to explicitly import in each file.

1.2 To make it easier to access the top level objects in our labs, we will define member variables to the keep the top level object accessible throughout this document or the class. Revit has a concept of separation between DB and UI objects. This applies to the Revit application and document objects.

These top level objects are accessible as follows:

- UIApplication ← commandData.Application
- Application ←UIApplication.Application
- UIDocument ← UIApplication.ActiveUIDocument
- Document ← UIDocument.Document

Define member variables, e.g., m_rvtApp and m_rvtDoc, to keep DB level application and document respectively. The following is an example:

```
<C#>
//  DB Element - learn about Revit element
[Transaction(TransactionMode.Manual)]
public class DBElement : IExternalCommand
{
    //  Member variables
    Application m_rvtApp;
    Document m_rvtDoc;

    public Result Execute(ExternalCommandData commandData,
                          ref string message,
                          ElementSet elements)
    {
        //  Get the access to the top most objects.
        UIApplication rvtUIApp = commandData.Application;
        UIDocument rvtUIDoc = rvtUIApp.ActiveUIDocument;
        m_rvtApp = rvtUIApp.Application;
        m_rvtDoc = rvtUIDoc.Document;

        //  ...

        return Result.Succeeded;
    }
}
</C#>
```

Note: about VB.NET vs. C# behavior difference
There is a behavior difference between VB.NET and C#; VB.NET automatically adds the root namespace, while C# does not.

For C# developer: To make the code easier to read in this instruction, we are omitting the namespace from now on. Please insert the code between the namespace as needed.

```
<C#>
namespace IntroCs
{
}
</C#>
```

Note: Using the member variable this way is just to make our life slightly easier during our exercises. You may find other approaches more suitable depending on the context of your program.

2. Pick an Element

Autodesk.Revit.DB.Element is a base class for objects in the Revit project database. We are going to pick an arbitrary element on the UI screen and examine it to learn more about elements in Revit.

We can use one of overloaded PickObject() method to pick an object on the screen:

- UIDocument.Selection.PickObject(ObjectType.Element, promptString)

(We'll come to the topic of UI and selection when we get into the UI portion of the training. For now, this should be enough for the purpose of this lab.)

The following code demonstrates the usage:

<C#>
```
        // (1) pick an object on a screen.

        Reference refPick = rvtUIDoc.Selection.PickObject(
            ObjectType.Element, "Pick an element");

        // we have picked something.
        Element elem = m_rvtDoc.GetElement(refPick);
```
</C#>

`PickObject`() returns Reference object. You can retrieve the Element from the reference object returned.

3. Basic Element Information

In typical programming or usage of APIs, we identify the given object by checking its class names. Does the same apply to Revit API? The exercise in this section will answer this question.

3.1 Write a function which takes Element as an argument, and display the following properties of the given element:

- Class name (or Type in .NET)
- Category name
- Id (Element Id)

Then do the same with the family type of the given element. To get to the family type of the given element, you can use Element.GetTypeId() to obtain its Id first, then use Document.GetElement(elementId).
Let's name this function, for example, **ShowBasicElementInfo()**. The following shows a sample code to do this:

<C#>

```
    public void ShowBasicElementInfo(Element elem)
    {
        // let's see what kind of element we got.
        //
        string s = "You Picked:" + "\n";

        s += " Class name = " + elem.GetType().Name + "\n";
        s += " Category = " + elem.Category.Name + "\n";
        s += " Element id = " + elem.Id.ToString() + "\n" + "\n";

        // and, check its type info.
        //
        //Dim elemType As ElementType = elem.ObjectType '' this is obsolete.
        ElementId elemTypeId = elem.GetTypeId();
        ElementType elemType = (ElementType)m_rvtDoc.GetElement(elemTypeId);

        s += "Its ElementType:" + "\n";
        s += " Class name = " + elemType.GetType().Name + "\n";
        s += " Category = " + elemType.Category.Name + "\n";
        s += " Element type id = " + elemType.Id.ToString() + "\n";

        // finally show it.

        TaskDialog.Show("Basic Element Info", s);
    }
```

</C#>

Then, call this function from your main Execute() method right after you have picked an element:

<C#>

```
        //' we have picked something.
        Element elem = refPick.Element;

        // (2) let's see what kind of element we got.
        ShowBasicElementInfo(elem);
```

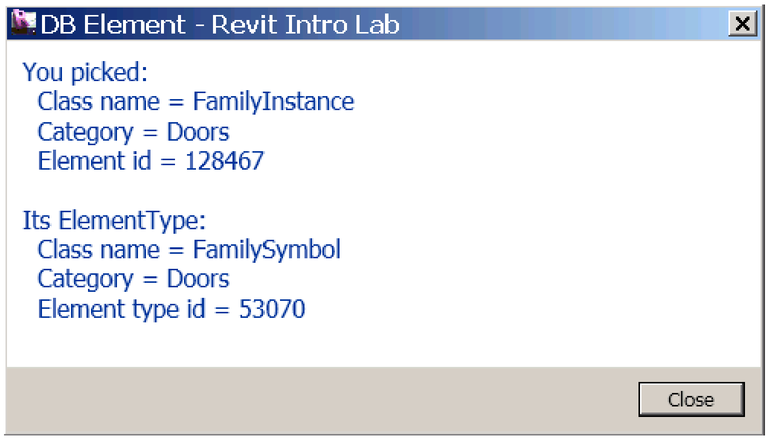
</C#>

3.2 Build the project. Add manifest file, e.g.:

```
  <AddIn Type="Command">
    <Text>DB Element</Text>
    <FullClassName>IntroCs.DBElement</FullClassName>
    <Assembly>C:\...\IntroCs.dll</Assembly>
```

```xml
    <AddInId>827AC040-6F44-4c03-82FE-292705580800</AddInId>
   <VendorId>ADNP</VendorId>
   <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
 </AddIn>
```

Run the command, "DB Element". Pick an element. You will see a dialog like following showing the class, category and id of the element that you just picked (Figure 3). Try picking a few other element and observe the output.



```
You picked:
  Class name = FamilyInstance
  Category = Doors
  Element id = 128467

Its ElementType:
  Class name = FamilySymbol
  Category = Doors
  Element type id = 53070
```

Figure3. Dialog showing the a few properties from a door.

Discussion:

- Compare the class names and categories among different elements, such as walls, doors and windows. What do you observe?
- Could you identify the element from the class name?
- Could you identify the element from the category?

Discuss these with your colleagues and instructor.

4. Identify Element

As you have found out by now, a class name is not enough to identify an element in Revit. Depending on an element you have, you will need to check the following:

- Class name
- Category
- If an element is Element Type (Symbol) or not

Table 1 shows the examples of a few elements with their class names and categories used to identify the element. They are divided into four areas: System Family vs. Component Family, and Family Type vs. Instance. This probably gives you slightly clear view of why you can identity some element by the class name and some requires category.

- A system family are a built-in object in Revit. There is a designated class for it. You can use it to identify the element.

- A component family has a generic form as FamilyInstance/FamilySymbol. Category is the way to further identify the kind of object it is representing in Revit.

	System Family	**Component Family**
Family Type	WallType FloorType	FamilySymbol & Category - Doors, Windows
Instance	Wall Floor	FamilyInstance & Category - Doors, Windows

Table 1. class names and categories that you can use to identify an element for walls, floors, doors and windows.

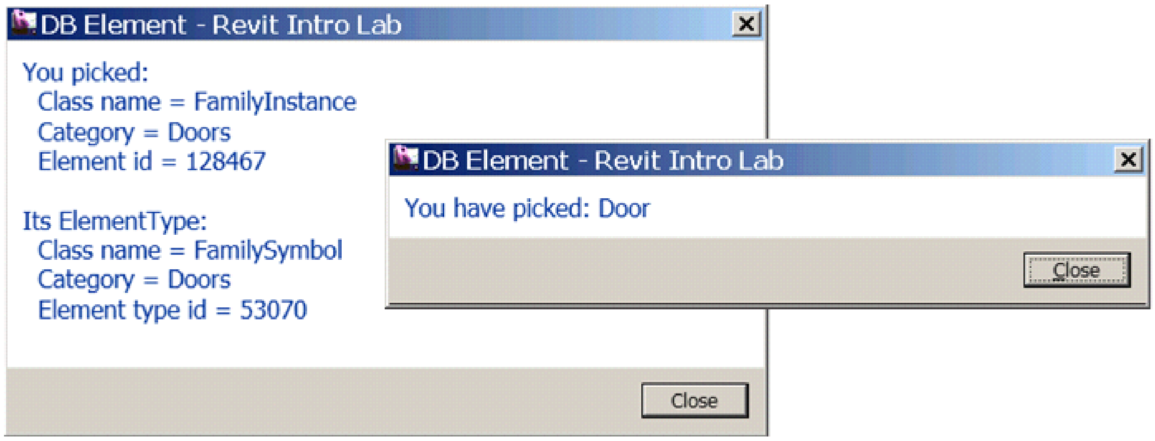
Now that we have understood how an element is represented in Revit database, let's add a code to identify an element. Here is an example:

<C#>
```
    // identify the type of the element known to the UI.
    public void IdentifyElement(Element elem)
    {

        // An instance of a system family has a designated class.
        // You can use it identify the type of element.
        // e.g., walls, floors, roofs.
        //
        string s = "";

        if (elem is Wall)
        {
            s = "Wall";
        }
        else if (elem is Floor)
        {
            s = "Floor";
        }
        else if (elem is RoofBase)
        {
            s = "Roof";
        }
        else if (elem is FamilyInstance)
        {
            // An instance of a component family is all FamilyInstance.
            // We'll need to further check its category.
            // e.g., Doors, Windows, Furnitures.
            if (elem.Category.Id.IntegerValue ==
                  (int)BuiltInCategory.OST_Doors)
            {
                s = "Door";
            }
            else if (elem.Category.Id.IntegerValue ==
                    (int)BuiltInCategory.OST_Windows)
            {
                s = "Window";
            }
            else if (elem.Category.Id.IntegerValue ==
                    (int)BuiltInCategory.OST_Furniture) {
                s = "Furniture";
            }
            else
            {
                // e.g. Plant
                s = "Component family instance";
            }
        }
        else if (elem is HostObject)
        {
            // check the base class. e.g., CeilingAndFloor.
            s = "System family instance";
        }
        else
        {
            s = "Other";
        }

        s = "You have picked: " + s;

        // show it.
        TaskDialog.Show("Identify Element", s);
    }
```
</C#>

Call this function from your main Execute() method right after ShowBasicElementInfo():

<C#>
```
        // (2) let's see what kind of element we got.
        ShowBasicElementInfo(elem);

        // (3) identify each major types of element.
        IdentifyElement(elem);
```
</C#>

Build and run the command “DB Element” once again to see if you can identify an element you have picked. Figure 4 shows a sample image of running the command.


DB Element - Revit Intro Lab

You picked:
  Class name = FamilyInstance
  Category = Doors
  Element id = 128467

Its ElementType:
  Class name = FamilySymbol
  Category = Doors
  Element type id = 53070

DB Element - Revit Intro Lab

You have picked: Door

Figure 4. Identifying an element

5. Parameters

Parameters property of an Element class largely corresponds to an element or family "properties" that you see in the UI. In the Revit API, there are four ways to access those properties or parameters:

- Element.get_Parameter– takes an argument that can identify the kind of parameter and returns the single parameter.
- Element.LookupParameter – returns a single parameter given the name.
- Element. GetOrderedParameters – returns a set of parameters applicable to the given element.
- Element.GetParameters – returns all parameters with the given name.

5.1 Retrieving a Set of Parameters through GetOrderedParameters ()

Let's first look at the GetOrderedParameters(). The code below demonstrates the usage. GetOrderedParameters() return a set of parameters. You can simply loop through it to access each parameter. The main part that you will need to pay attention is that that you will need to parse each parameter by the StorageType; a parameter can be Integer, Double, String and ElementId. Depending on the StorageType, you will need to choose the method to get the actual value.

<C#>

```
    // show all the parameter values of the element
    public void ShowParameters(Element elem, string header)
    {

        ParameterSet paramSet = elem.GetOrderedParameters();
        string s = string.Empty;

        foreach (Parameter param in paramSet)
        {
            string name = param.Definition.Name;
            // see the helper function below
            string val = ParameterToString(param);

            s += name + " = " + val + "\n";
        }


        TaskDialog.Show(header, s);
    }

    // Helper function: return a string from of the given parameter.
    //
    public static string ParameterToString(Parameter param)
    {

        string val = "none";

        if (param == null)
        {
            return val;
        }

        // to get to the parameter value, we need to pause it depending
        // on its strage type
        switch (param.StorageType)
        {
            case StorageType.Double:
                double dVal = param.AsDouble();
                val = dVal.ToString();
                break;

            case StorageType.Integer:
                int iVal = param.AsInteger();
                val = iVal.ToString();
                break;

            case StorageType.String:
                string sVal = param.AsString();
                val = sVal;
                break;

            case StorageType.ElementId:
                ElementId idVal = param.AsElementId();
                val = idVal.IntegerValue.ToString();
                break;

            case StorageType.None:
                break;

            default:
                break;
        }

        return val;
    }
```
</C#>

Call this function from your main Execute() method after IdentifyElement(). You may also use the same function to display its family type information.

```
<C#>
        // (3) identify each major types of element.
        IdentifyElement(elem);

        // (4) first parameters.
        ShowParameters(elem, "Element Parameters");

        //  check to see its type parameter as well
        //
        ElementId elemTypeId = elem.GetTypeId();
        ElementType elemType = (ElementType)m_rvtDoc.GetElement(elemTypeId);
        ShowParameters(elemType, "Type Parameters");
</C#>
```

Build and run the command “DB Element” once again. You should be a list of parameters displayed in dialogs. Figure 2 (on page 2) shows a sample image of running the command.

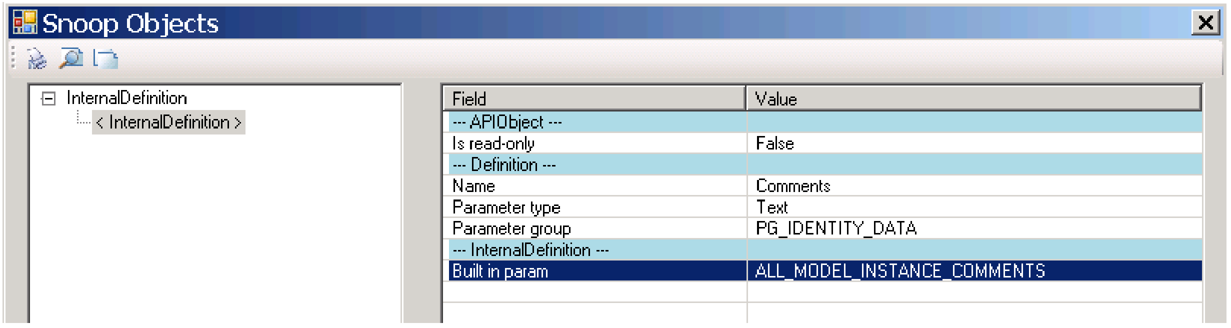
5.2 Retrieving an Individual Parameter Using BuiltInParameter

There are four ways to access individual parameters:

- Parameter(**BuiltInParameter**) – retrieve a parameter using a parameter Id.
- LookupParameter(String) – retrieve using the name.
- Parameter(Definition) – retrieve from its definition.
- Parameter(GUID) – retrieve shared parameter using GUID for a shared parameter.

Here we will take a look at the first and second. Calling LookupParameter(Xxx) method itself using the name is straightforward. However, using name has a disadvantage of depending on a language version of Revit you are running. Therefore, using BuiltInParameter is more ideal. The trick here is to find out which parameter Id or BuiltInParameter to use. If you look at the RevitAPI.chm documentation, under BuiltInParameter Enum section, you will see hundreds of BuiltinParameters defined. Among those BuiltInParameters, only fraction of enum are applicable to a given element. How can we find out which BuiltInParameter to use to retrieve a specific parameter?

RevitLookup tool comes handy to explore and find out which BuiltInParameter corresponds to which parameter name. When you want to find out a parameter for a specific type of element, simple click on the same type of object in the project >> [Snoop Current Selection …] >> [Parameters]. When you click on each parameter name, you can check its [Definition] to find out the corresponding BuiltInParameter (Figure 5).



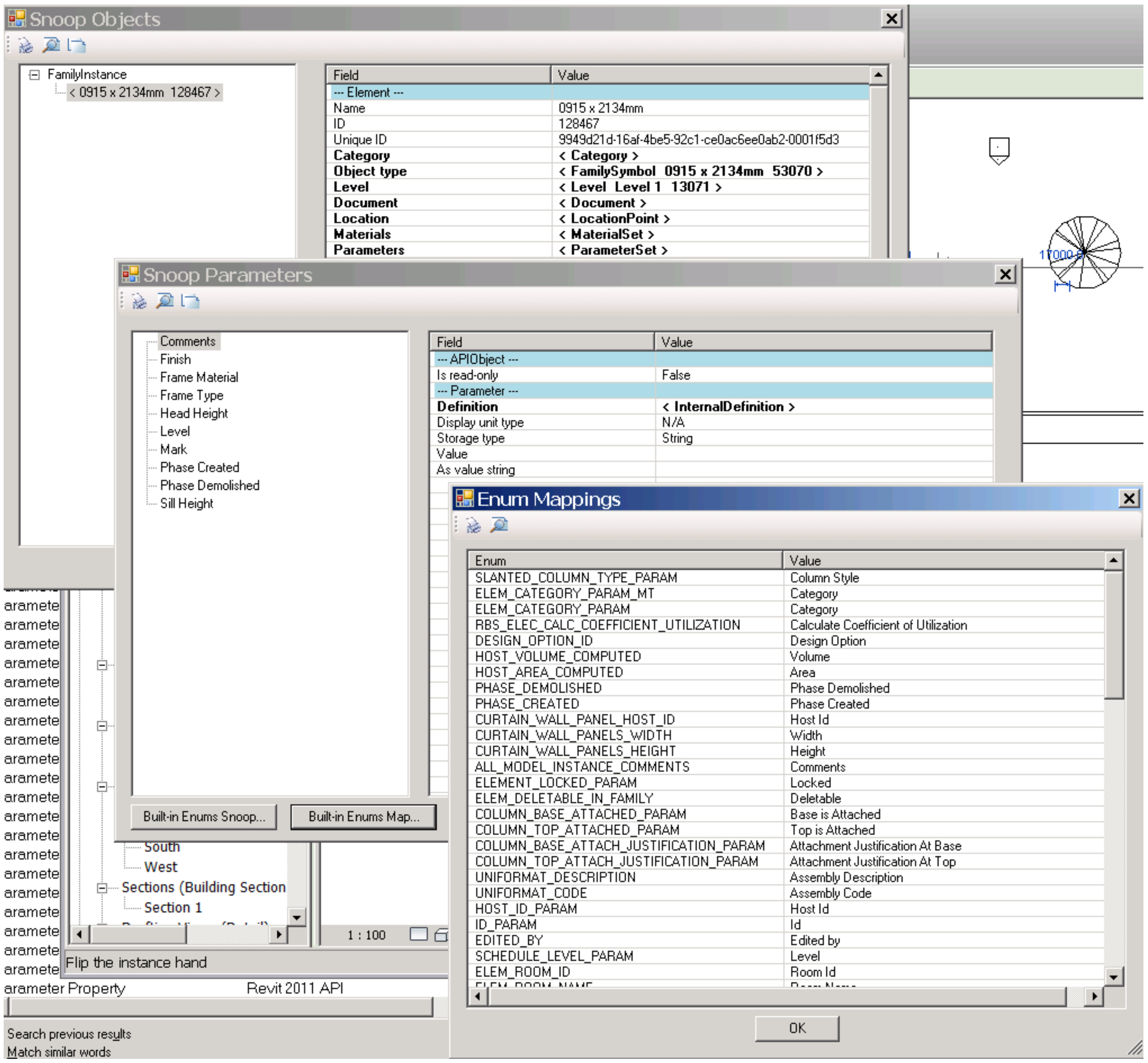
Field	Value
--- APIObject ---	
Is read-only	False
--- Definition ---	
Name	Comments
Parameter type	Text
Parameter group	PG_IDENTITY_DATA
--- InternalDefinition ---	
Built in param	ALL_MODEL_INSTANCE_COMMENTS

Figure 5. Use RevitLookup to check the definition of parameter.

Another thing that may worth mentioning is that if you look at the list of BuiltInParameters, using [Built-in Enums Snoop…] or [Built-in Enumes Map…] button, you will see more parameters than you see in parameters list. Occasionally, you may find some useful properties there. Such a sample will be:

- BuiltInParameter.SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM – family and type name
- BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM – family name

You can use these to retrieve family and type names of the given element. Figure 6 shows a sample enum mapping.



Snoop Objects

FamilyInstance
< 0915 x 2134mm 128467 >

Field	Value
--- Element ---	
Name	0915 x 2134mm
ID	128467
Unique ID	9949d21d-16af-4be5-92c1-ce0ac6ee0ab2-0001f5d3
Category	**< Category >**
Object type	**< FamilySymbol 0915 x 2134mm 53070 >**
Level	**< Level Level 1 13071 >**
Document	**< Document >**
Location	**< LocationPoint >**
Materials	**< MaterialSet >**
Parameters	**< ParameterSet >**

Snoop Parameters

- Comments
- Finish
- Frame Material
- Frame Type
- Head Height
- Level
- Mark
- Phase Created
- Phase Demolished
- Sill Height

Field	Value
--- APIObject ---	
Is read-only	False
--- Parameter ---	
Definition	**< InternalDefinition >**
Display unit type	N/A
Storage type	String
Value	
As value string	

Built-in Enums Snoop... Built-in Enums Map...

Enum Mappings

Enum	Value
SLANTED_COLUMN_TYPE_PARAM	Column Style
ELEM_CATEGORY_PARAM_MT	Category
ELEM_CATEGORY_PARAM	Category
RBS_ELEC_CALC_COEFFICIENT_UTILIZATION	Calculate Coefficient of Utilization
DESIGN_OPTION_ID	Design Option
HOST_VOLUME_COMPUTED	Volume
HOST_AREA_COMPUTED	Area
PHASE_DEMOLISHED	Phase Demolished
PHASE_CREATED	Phase Created
CURTAIN_WALL_PANEL_HOST_ID	Host Id
CURTAIN_WALL_PANELS_WIDTH	Width
CURTAIN_WALL_PANELS_HEIGHT	Height
ALL_MODEL_INSTANCE_COMMENTS	Comments
ELEMENT_LOCKED_PARAM	Locked
ELEM_DELETABLE_IN_FAMILY	Deletable
COLUMN_BASE_ATTACHED_PARAM	Base is Attached
COLUMN_TOP_ATTACHED_PARAM	Top is Attached
COLUMN_BASE_ATTACH_JUSTIFICATION_PARAM	Attachment Justification At Base
COLUMN_TOP_ATTACH_JUSTIFICATION_PARAM	Attachment Justification At Top
UNIFORMAT_DESCRIPTION	Assembly Description
UNIFORMAT_CODE	Assembly Code
HOST_ID_PARAM	Host Id
ID_PARAM	Id
EDITED_BY	Edited by
SCHEDULE_LEVEL_PARAM	Level
ELEM_ROOM_ID	Room Id

OK

South
West
Sections (Building Section
Section 1
1 : 100
Flip the instance hand
arameter Property Revit 2011 API
Search previous results
Match similar words

Figure 6. you can use RevitLookup to explore which BuiltInParameter to use.

Below is a sampler that shows how to retrieve some of commonly used properties. Explore the BuiltInParameters in RevitLookup and try writing a function to retrieve a few parameters of your interests using the code below as an example.

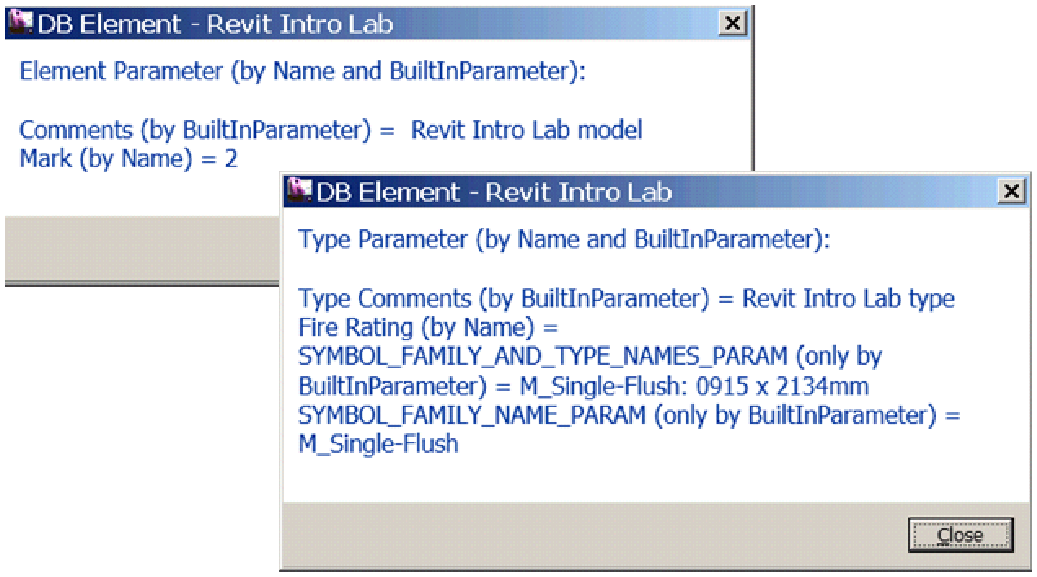
<C#>

```
    //  example of retrieving a specific parameter indivisually.
    //  (hard coding for simplicity. This function works best
    //  with walls and doors.)

    public void RetrieveParameter(Element elem, string header)
    {
        string s = string.Empty;

        // as an experiment, let's pick up some arbitrary parameters.
        // comments - most of instance has this parameter

        // (1) by BuiltInParameter.
        Parameter param =
            elem.get_Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS);
        if (param != null)
        {
            s += "Comments (by BuiltInParameter) = " +
                ParameterToString(param) + "\n";
        }

        // (2) by name. (Mark - most of instance has this parameter.)
        // if you use this method, it will language specific.
        param = elem.LookupParameter("Mark");
        if (param != null)
        {
            s += "Mark (by Name) = " + ParameterToString(param) + "\n";
        }

        // the following should be in most of type parameter
        //
        param = elem.get_Parameter(BuiltInParameter.ALL_MODEL_TYPE_COMMENTS);
        if (param != null)
        {
            s += "Type Comments (by BuiltInParameter) = " +
                ParameterToString(param) + "\n";
        }

        param = elem.LookupParameter("Fire Rating");
        if (param != null)
        {
            s += "Fire Rating (by Name) = " + ParameterToString(param) +
                "\n";
        }

        // using the BuiltInParameter, you can sometimes access one that is
        // not in the parameters set.
        // Note: this works only for element type.

        param = elem.get_Parameter(
            BuiltInParameter.SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM);
        if (param != null)
        {
            s += "SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM (only by
BuiltInParameter) = " +
                  ParameterToString(param) + "\n";
        }

        param = elem.get_Parameter(BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM);
        if (param != null)
        {
          s += "SYMBOL_FAMILY_NAME_PARAM (only by BuiltInParameter) = "
              + ParameterToString(param) + "\n";
        }
        // show it.

        TaskDialog.Show(header, s);
    }
</C#>
```

Call your function from your main `Execute()` method at the end. You may also use the same function to display its family type information.

```
<C#>
        // (4) first parameters.
        ShowParameters(elem, "Element Parameters: ");

        //  check to see its type parameter as well
        //
        ElementId elemTypeId = elem.GetTypeId();
        ElementType elemType = (ElementType)m_rvtDoc.GetElement(elemTypeId);
        ShowParameters(elemType, "Type Parameters: ");

        // access to each parameters.
        RetrieveParameter(
            elem, "Element Parameter (by Name and BuiltInParameter)");
        // the same logic applies to the type parameter.
        RetrieveParameter(
            elemType, "Type Parameter (by Name and BuiltInParameter)");

</C#>
```

Build and run the command “DB Element” once again. You should see a list of parameters of your choice displayed in dialogs (Figure7).



DB Element - Revit Intro Lab

Element Parameter (by Name and BuiltInParameter):

Comments (by BuiltInParameter) = Revit Intro Lab model
Mark (by Name) = 2

DB Element - Revit Intro Lab

Type Parameter (by Name and BuiltInParameter):

Type Comments (by BuiltInParameter) = Revit Intro Lab type
Fire Rating (by Name) =
SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM (only by BuiltInParameter) = M_Single-Flush: 0915 x 2134mm
SYMBOL_FAMILY_NAME_PARAM (only by BuiltInParameter) = M_Single-Flush

Figure 7. You can use BuiltInParameters to access parameters individually.

6. Location Information

Location of each element is stored under Location property. A location can be point-based (LocationPoint) or curve/line-based (LocationCurve). You will need to cast to LocationPoint or LocationCurve in order to access more properties. The following demonstrates the usage:

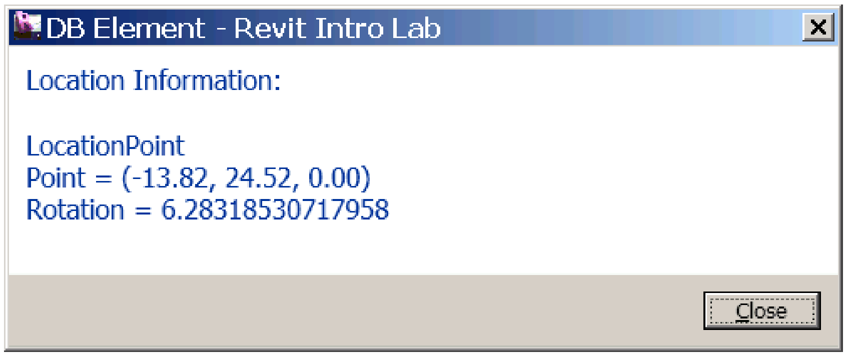
<C#>

```
    // show the location information of the given element.
    // location can be LocationPoint (e.g., furniture), and LocationCurve
    // (e.g., wall).

    public void ShowLocation(Element elem)
    {

        string s = "Location Information: " + "\n" + "\n";
        Location loc = elem.Location;

        if (loc is LocationPoint)
        {
            // (1) we have a location point

            LocationPoint locPoint = (LocationPoint)loc;
            XYZ pt = locPoint.Point;
            double r = locPoint.Rotation;

            s += "LocationPoint" + "\n";
            s += "Point = " + PointToString(pt) + "\n";
            s += "Rotation = " + r.ToString() + "\n";
        }
        else if (loc is LocationCurve)
        {
            // (2) we have a location curve

            LocationCurve locCurve = (LocationCurve)loc;
            Curve crv = locCurve.Curve;

            s += "LocationCurve" + "\n";
            s += "EndPoint(0)/Start Point = " +
                  PointToString(crv.GetEndPoint(0)) + "\n";
            s += "EndPoint(1)/End point = " +
                  PointToString(crv.GetEndPoint(1)) + "\n";
            s += "Length = " + crv.Length.ToString() + "\n";

           // Location Curve also has property JoinType at the end

           s += "JoinType(0) = " + locCurve.get_JoinType(0).ToString() + "\n";
           s += "JoinType(1) = " + locCurve.get_JoinType(1).ToString() + "\n";

        }
        // show it.

        TaskDialog.Show("Show Location", s);
    }


    // Helper Function: returns XYZ in a string form.
    //
    public static string PointToString(XYZ pt)
    {
        if (pt == null)
        {
            return "";
        }

        return string.Format("({0},{1},{2})",
            pt.X.ToString("F2"), pt.Y.ToString("F2"), pt.Z.ToString("F2"));
    }
</C#>
```

Call your function from your main `Execute()` method at the end.

```
<C#>
        // (4) first parameters.
        ...
        // the same logic applies to the type parameter.
        RetrieveParameter(elemType, "Type Parameter (by Name and
BuiltInParameter): ");

        // (5) location
        ShowLocation(elem);

</C#>
```

Build and run the command “DB Element” once again. You should see location information displayed in dialogs. Figure 8 shows an example of location information when you pick an element based on Location Point, such as a door. When you pick a wall, it will be Location Line. Note, not all elements allow accessing location information this way.


DB Element - Revit Intro Lab

Location Information:

LocationPoint
Point = (-13.82, 24.52, 0.00)
Rotation = 6.28318530717958

Figure 8. Sample location info.

7. Geometry Information (Optional)

The last piece of element properties that we would like to take a look is Geometry. Writing a code for retrieving Geometry information can get a little involved and beyond the scope of this training. We'll only describe a big picture here.

- Geometry Options – you can specify the detail level (Fine, Medium and Fine) when you retrieve geometry information from an element.
- A Geometry object can be Solid, Geometry Instance, Curve or Mesh. Geometry Instance is an instance of another element (symbol), such as a window and a door.

The code below shows the access to the high level geometry representation. For further drill down of Solids/Faces/Edges, please use RevitLookup. RevitCommands sample has a simple example. The following SDK samples show other geometry access with a little viewer:

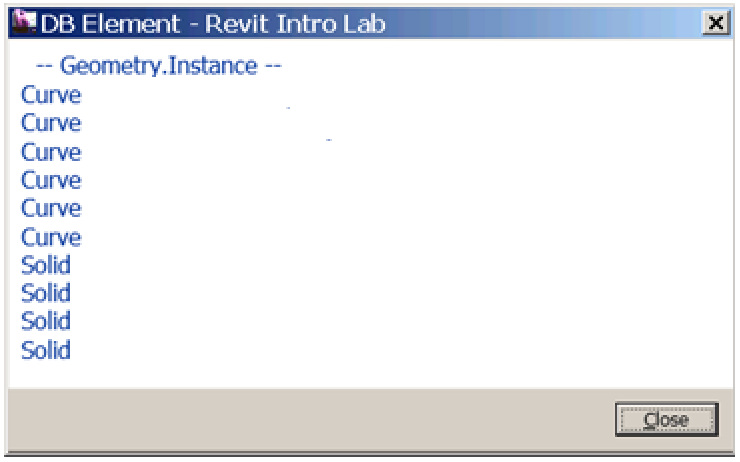
- ElementViewer
- RoomViewer
- AnalyticalViewer

<C#>
```
    // show the geometry information of the given element.
    public void ShowGeometry(Element elem)
    {
        // Set a geometry option
        Options opt = m_rvtApp.Create.NewGeometryOptions();
        opt.DetailLevel = ViewDetailLevel.Fine;

        // Get the geometry from the element
        GeometryElement geomElem = elem.get_Geometry(opt);

        // If there is a geometry data, retrieve it as a string to show it.
        string s = (geomElem == null) ?
          "no data" :
          GeometryElementToString(geomElem);

        TaskDialog.Show("Show Geometry", s);
    }


    // Helper Function: parse the geometry element by geometry type.
    // Here we look at the top level.

    public static string GeometryElementToString(GeometryElement geomElem)
    {
        string str = string.Empty;

        foreach (GeometryObject geomObj in geomElem)
        {
            if (geomObj is Solid)
            {
                // ex. wall
                Solid solid = (Solid)geomObj;
                //str += GeometrySolidToString(solid)

                str += "Solid" + "\n";
            }
            else if (geomObj is GeometryInstance)
            {
                // ex. door/window

                str += " -- Geometry.Instance -- " + "\n";
                GeometryInstance geomInstance = (GeometryInstance)geomObj;
                GeometryElement geoElem = geomInstance.SymbolGeometry;

                str += GeometryElementToString(geoElem);
            }
            else if (geomObj is Curve)
            {
                Curve curv = (Curve)geomObj;
                //str += GeometryCurveToString(curv)

                str += "Curve" + "\n";
            }
            else if (geomObj is Mesh)
            {
                Mesh mesh = (Mesh)geomObj;
                //str += GeometryMeshToString(mesh)

                str += "Mesh" + "\n";
            }
            else
            {
                str += " *** unkown geometry type" +
                    geomObj.GetType().Name;
            }
        }

        return str;
    }
</C#>
```

Call your function from the end of your main `Execute()` method.

```
<C#>
        // (5) location
        ShowLocation(elem);

        // (6) geometry - the last piece. (Optional)
        ShowGeometry(elem);
</C#>
```

Build and run the command “DB Element” once again. You should be geometry information displayed in dialogs. (Figure 9).



```
DB Element - Revit Intro Lab
  -- Geometry.Instance --
Curve
Curve
Curve
Curve
Curve
Curve
Solid
Solid
Solid
Solid
```

Figure 9. an example of high level geometry information

8. Summary

In this lab, we have learned how an element is represented in Revit and how to retrieve information about an element. We have learned how to:

- Identify an element using class name, category, and symbol or not.
- Retrieve a set of properties of an element using Parameters()
- Retrieve a specific property of an element using BuiltInParameter
- Retrieve location information
- Retrieve graphic information

In the next lab, we will take a look at a group of elements in the Revit database and learn how to selectively retrieve elements of our interests, which is called element filtering.